PLACCEA: Plaforma para Creación y Corrección de Ejercicos Académicos

Proyecto Sistemas Informáticos

Sergio Barja Valdés,
Pablo Arteaga Álvarez e
Ignacio Moya Señas
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Director: César Andrés Sánchez
Director: Luis Llana Díaz

Autorización de difusión

Los abajo firmantes Ignacio Moya Señas, Pablo Arteaga Álvarez y Sergio Barja Valdés, matriculados en SISTEMAS INFORMÁTICOS de la Facultad de Informática, autorizamos a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo: "Plaforma para Creación y Corrección de Ejercicos Académicos", realizado durante el curso académico 2011-2012 bajo la dirección de los profesores César Andrés Sánchez y Luis Llana Díaz en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Ignacio Moya Señas	Pablo Arteaga Álvarez	Sergio Barja Valdés
11857158-Z	51076849-J	51064753-S

En Madrid a 18 de junio de 2012

Agradecimientos

A nuestras familias por su apoyo y cariño durante el largo año que hemos dedicado a este proyecto. A nuestro amigos, por darnos la posibilidad de evadirnos y ayudarnos a desconectar. A nuestros profesores, que unos casos han entendido que no solo desarrollamos un proyecto, y en otros entendían que nuestro trabajo no se limitaba sólamente a su asignatura.

A los empleados de los laboratorios y de la biblioteca, pues con su comprensión, con sus buenas maneras y con su falta de servicios mínimos durante las huelgas, movieron nuestro centro de estudio a las mesas de trabajo en grupo de los pasillos.

Al lector, pues deseamos que la experiencia de este texto le resulte lo mejor posible.

A todos, gracias.

Resumen

Nuestro proyecto tiene como finalidad el desarrollo de una aplicación que constituya una plataforma web de prácticas de programación en Java. El objetivo final es doble. Por un lado *facilitar el acceso* a estos enunciados por parte de los alumnos así como la *entrega de las resoluciones* de las prácticas propuestas. Por otro realizar una serie de pruebas de manera automática sobre los programas entregados por los alumnos que proporcionen información valiosa para la corrección de las prácticas y su calificación.

Los alumnos podrán consultar en la plataforma web los enunciados de los distintos ejercicios prácticos de la asignatura y podrán entregar dichos ejercicios resueltos, subiendo el código de sus programas desde el navegador web. Cada ejercicio consistirá en el desarrollo de un programa (en Java) que resuelva un problema especificado en el enunciado del mismo.

Estos enunciados podrán ser creados por los profesores e incorporados a la plataforma también desde el navegador web. Un profesor podrá definir el enunciado de un nuevo ejercicio práctico así como proporcionar otro tipo de información necesaria para la elaboración de las distintas pruebas, como por ejemplo el código del programa que implementa una solución óptima al problema planteado. También podrán consultar los ejercicios entregados por sus alumnos así como los resultados y estadísticas obtenidos tras la ejecución de las pruebas pertinentes. Para cada enunciado se podrá escoger un conjunto de pruebas a realizar de entre las que el sistema ofrezca. Contará con pruebas de estructura, robustez, corrección y eficiencia. A lo largo de este documento se presentará de una manera detallada el sistema informático construido para alcanzar el objetivo aquí planteado.

Palabras clave

E-learning.

Reflexión.

Java.

PHP.

Abstract

Our project aims to achive the development of a web platform for Java programming exercises. We have double goal: help student access to exercise's statement and result delivering, and the execution of automatic test series over the exercices delivered for the students that provide some value information for exercise's correction and calification.

Students will be able to consult the exercise's statements of different practice task from the subject and deliver that task's solution, uploading their source code using web browser. Each task consist in the development of the code (in Java) that resolves the specifict problem from the statement.

These statements would be created by the teachers and uploaded to the platform with web browser too. Teachers would define the statement of a new practice task and provide any kind of information needed for the development of the distint test, for example, the code that implements an optimal solution to the given problem. Teachers also would consult the exercises uploaded by the students, their results and statics obtained from executing the required tests.

Each statement would be chosen for a set of test from any the system could offer. These test will contain structure, sturdiness, correction and performance tests.

Along this paper we will present in a detailed way the computer system built to achieve the goal presented here.

Keywords

E-learning.
Reflection.
Java.
PHP.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 1

Introducción

A lo largo de este apartado se presenta una visión global del proyecto y las motivaciones que han llevado a su desarrollo. En primer lugar se expone el estado del arte del E-learning. En segundo lugar se expone la motivación del proyecto, esto es, se presentan las necesidades que otros sistemas actuales no cubren y que nuestro sistema sí cubrirá. Para terminar, se expone qué se espera aportar.

La estructura de este apartado es la siguiente:

- 1.1 **Estado del arte**. E-learning.
- 1.2 **Tecnologías y sistemas actuales**. Presentación de soluciones actuales.
- 1.3 **Motivación y necesidad**. ¿Por qué las soluciones actuales no satisfacen nuestras necesidades?

1.1. Estado del arte

El término E-learning se aplica al desarrollo de la educación a distancia a través de diversos canales electrónicos. Estos canales pueden ser correos electrónicos, foros de discusión, mensajería instantánea, páginas web y plataformas de formación; aunque es en este último en el que nos vemos más interesados. Desafortunadamente, la industria no nos brinda una definición más precisa.

Esta definición tan vaga, ha llevado a una situación donde llamaremos E-learning a todo aprendizaje por vía tecnológica. Como consecuencia, le otorga una dimensión enorme. Esta dimensión ha llevado a separarlo en tres enfoques diferentes:

- **Computer Assisted Instruction (CAI)** [3, 1, 14]. También llamado Computer-based training, se refiere al sector del E-learning que proporciona enseñanza convencional (tutoriales, teoría o ejercicios) sobre un medio electrónico.
- **Computer-Managed Instruction (CMI)** [19]. Abarca los productos que proporcionan conocimientos a adquirir, recursos para adquirirlos y asistencia en el rendimiento del estudiante mediante canales electrónicos.
- **Computer Supported Learning Resources (CSLR)** [16, 11]. Cubre los aspectos del E-learning que dan acceso a la información que el estudiante necesita para su proceso de estudio.

Podemos ver que desde la perspectiva de su concepción y desarrollo como herramienta, los sistemas de e-learning tienen una dualidad pedagógica y tecnológica. Pedagógica porque a estos sistemas deben hacer todo lo posible por no ser meros contenedores de información en formato digital, sino que debe seguir unos modelos y patrones pedagógicamente definidos, dado que aparecen nuevos contextos. Tecnológica porque como ya hemos comentado, la experiencia educativa se desarrolla sobre canales electrónicos. En concreto, aplicaciones desarrolladas en ambientes web, que proporcionan el sobrenombre de plataformas de formación.

Es este tipo de aplicaciones el que algunas veces se denomina E-learning asíncrono [2]. En el E-learning asíncrono, la interacción entre el profesor y el alumno no coincide ni en el espacio ni en el tiempo. Esta forma es la más flexible para el alumno, pues el papel del alumno es mucho más autónomo.

Existe también el término B-learning (Blended learning) [15, 7, 18] donde la enseñanza a distancia desarrollada sobre la plataforma se ve complementada con enseñanza presencial. En el caso de nuestro proyecto, donde presentamos una plataforma de apoyo a una clase presencial, podríamos decir que estamos haciendo B-learning.

En definitiva, el E-learning proporciona un acceso fácil y en ocasiones a bajo coste, al conocimiento y a las oportunidades de aprendizaje en general. Debido a esto, se ha producido una proliferación de la formación on-line y de las comunidades de aprendizaje.

1.2. Tecnologías y sistemas actuales

A continuación proponemos un repaso de las tecnologías y sistemas actuales desde el prisma de las plataformas educativas. Las plataformas educativas son fundamentales en los entornos virtuales de aprendizaje y enseñanza, conocidos como EVA [13].

Como es lógico, actualmente la mayoría de universidades de todo el mundo cuentan con una plataforma de este tipo, que facilite la consulta de materiales educativos, examenes en línea, sistemas de foros, avisos, entrega de tareas e incluso un correo interno. Además estas herramientas deben exibir una usabilidad suficiente para que los profesores de las distintas asignaturas puedan explotar estas características lo mejor posible.

Moodle Moodle [8], acronimo de Modular Object-Oriented Dynamic Learning Environment, es un Sistema de Gestión de Cursos de Código Abierto (Open Source Course Management System, CMS), conocido también como Sistema de Gestión del Aprendizaje (Learning Management System, LMS) o como Entorno de Aprendizaje Virtual (Virtual Learning Environment, VLE). Es una aplicación web gratuita que los educadores pueden utilizar para crear sitios de aprendizaje efectivo en línea.

De esta forma, es un paquete software para la creación de cursos en línea. Es un proyecto de desarrollo global destinado al soporte de un framework para uso educativo. Está distribuído como software libre (bajo licencia GPL). En la actualidad Moodle se ha extendido con múltiples módulos, donde destacamos éste [4] cuya finalidad era permitir a los alumnos cooperar entre ellos en el proceso de aprendizaje.

Sakai Sakai [12] es una comunidad para mejorar la enseñanza, el aprendizaje y la investigación, que trabaja en conjunto para definir las necesidades académicas de los usuarios, creando herramientas de software, compartiendo las mejores practicas, conocimientos y recursos para el apoyo de este objetivo.

Esta comunidad presenta la plataforma Sakai CLE (Colaboración y entorno de aprendizaje). Sakai CLE proporciona un entorno para la enseñanza y el aprendizaje basado en la colaboración y el conocimiento compartido.

Sakai se distribuye como software libre, aunque también existen suministradores comerciales que ofrecen los servicios de Sakai.

Helvia La plataforma educativa Helvia [6] está desarrollada en software libre para la Consejería de Educación de la Junta de Andalucía.

1.3. Motivación y necesidad

Como ya hemos visto, las tecnologías existentes proporcionan una gran capacidad de gestión de datos académicos. Para el alumno es importante tener una plataforma con la que acceder a los contenidos del curso y de las calificaciones.

Sin embargo, en nuestra opinión, sería conveniente que esta plataforma sirviera como canal para la corrección de ejercicios. Que esa corrección, además, fuera automática y no dependiente de interfaz humano.

Si el alumno tiene la capacidad de evaluar si su tarea es aceptable, correcta u óptima, desde su entorno de estudio; eliminaríamos el cuello de botella que produce la entrega personal tradicional, en entrevista con el profesor, o incluso compensamos la ausencia de ella. En ocasiones, como alumnos que somos, nos hemos encontrado en la dificil situación de descubrir que nuestro código (desde nuestro punto de vista, correcto y funcional) no cumple las exigencias del profesor. Si limitamos estas exigencias a una serie de pruebas, que podrían ser predefinidas o incluso propias, mejoramos la experiencia de estudio, haciéndolo más eficiente en tiempo y en esfuerzo.

Desde el enfoque del profesor, resultaría más cómodo definir qué pruebas tiene que cumplir una tarea correcta, siendo más dinámica la corrección.

CAPÍTULO 2

Objetivos

Cuando se nos propuso este proyecto, se nos plantearon una serie de objetivos que servirían de guía para nuestro trabajo. Estos objetivos no pretendían ser una lista concreta de requisitos que se debieran cumplir estrictamente a la finalización del proyecto. Más bien, planteaban una visión global de lo que se intentaba obtener, ofreciendo ideas sobre cómo abordar el problema planteado de una forma general y ambiciosa. De esta forma, no se restringiría el marco del mismo, y seríamos nosotros y nuestras limitaciones de tiempo los que constreñiríamos las fronteras de los logros que finalmente se alcanzarían.

1. Crear una herramienta para testear las prácticas de programación de la asignatura Tecnología de la Programación.
2. Gestionar alumnos y profesores en una base de datos.
3. Guardar información estadística de los alumnos.
4. Recuperar información de los alumnos, procesarla, y dar resultados no triviales de la misma.

CAPÍTULO 3

Resolutor de Ejercicios

Con el fin de atender nuestras necesidades a la hora de testear los ejercicios de los alumnos ponemos especial énfasis en la funcionalidad del *servidor*. El servidor se ocupará por un lado de ofrecer acceso a profesores y alumnos y por otro procesar peticiones, y guardar y procesar los resultados. Para procesar las pruebas de los alumnos, el servidor se servirá de un motor de pruebas.

Este motor está implementado en el lenguaje de programación Java. La interacción de un alumno con el motor la realizamos mediante una plataforma web y una base de datos. Podemos encontrar un esquema de la interacción alumno-motor se presenta en la Figura 3.1. En este esquema:

- `Usuario` respresenta al usuario que desea interaccionar con la apliación. Los usuarios que consideramos son los profesores y los alumnos. Para cada usuario consideramos diagramas de secuencia distintos. En la figura 3.3 mostramos el diagrama de secuencia de la interacción del profesor con la aplicación, y en la figura 3.2 mostramos la interacción del alumno.

- `Plataforma Web` respresenta la plataforma web, con la cual interactúa el alumno. El servidor procesará la petición escribiendo el correspondiente registro en la base de datos. Además despertará al motor en caso de que no estuviera activo. La implementación web del servidor se realiza en PHP y podemos encontrarla en el capítulo 4.

- `Motor Java` respresenta la implementación del motor de pruebas, que se encarga de

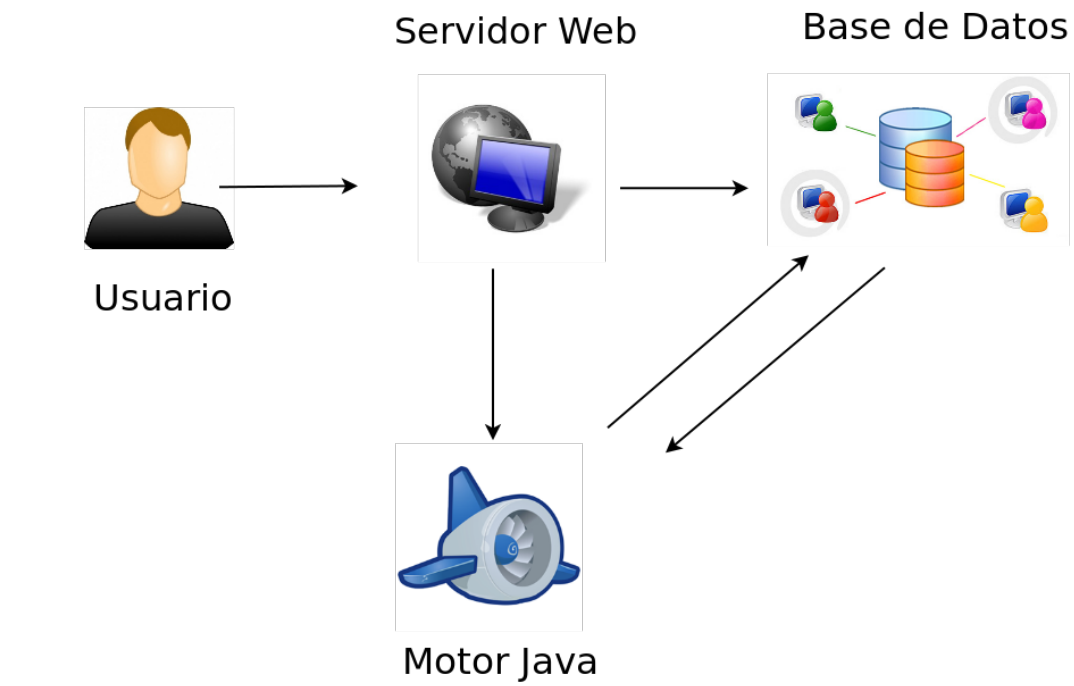


Figura 3.1: Esquema de la interacción alumno-motor.

ejecutar las pruebas solicitadas sobre el código entregado. En la Figura 3.4 se muestran los principales conjuntos de pruebas que se implementan en el motor, y que se recogen en el framework que presentamos en la sección 3.1. Las pruebas se implementan usando reflexión [5, 17]. Se consideran estos conjuntos buscando que sea incremental, es decir, que un alumno intente superar las pruebas de un conjunto antes de intentar superar el siguiente.

- `Base de Datos` respresenta la implementación en MySQL [9] del almacenamiento de la información de alumnos, profesores, asignaturas, ejercicios, pruebas y resultados. La interacción del motor con la base de datos la realizamos mediante una cola de peticiones. El diseño completo de la base de datos, así como una descripción detallada del mismo, podemos encontrarlo en la sección 4.4.2.

Una vez presentada la estructura general de nuestro servidor, vamos a centrarnos en cómo utilizar el motor Java. Cuando iniciamos el motor, éste se conecta a la base de datos para comprobar si existen peticiones pendientes.

Ejemplo 3.1 En nuestra aplicación una petición pendiente aparece representada como un nuevo registro en la tabla `cola` de la base de datos. En dicho registro guardaremos diversas

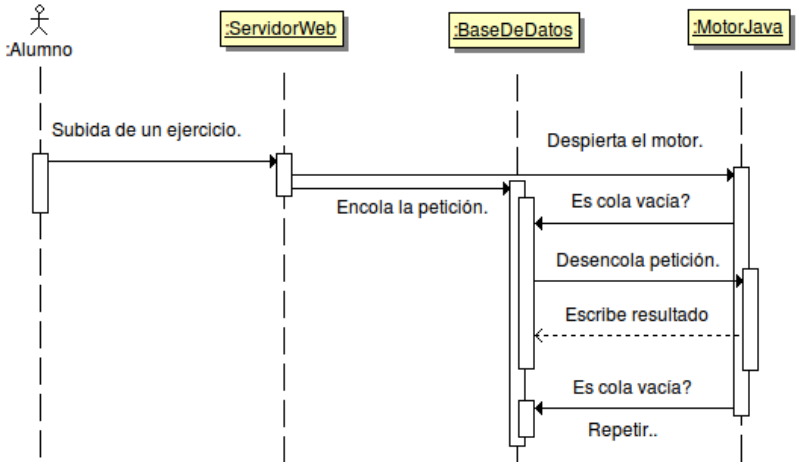


Figura 3.2: Diagrama de secuencia de la interacción del alumno con la aplicación.

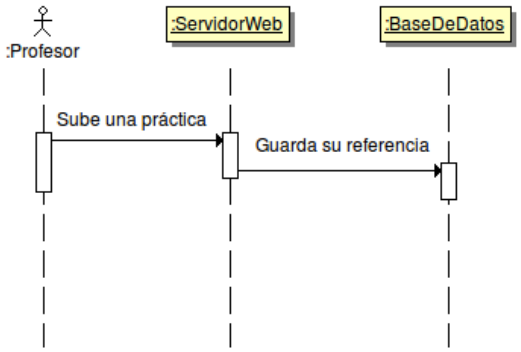


Figura 3.3: Diagrama de secuencia de la interacción del profesor con la aplicación.

entradas, como la tarea que desamos corregir, la referencia del profesor, el intento actual, y si ya ha sido atendida o no. Este registro y el resto de la base de datos se describen con mayor detalle en las secciones 4.4.2 y 4.5.3.

id	tarea	referencia	intento	esRef	atendida
1	1	1	0	1	0
2	1	1	1	0	0
3	1	1	2	0	0

En el ejemplo podemos ver 3 peticiones no atendidas. Todas referentes a la misma tarea y con la misma referencia. Con los datos del registro `cola` recogemos la información necesaria para procesar la petición.

El resultado lo guardaremos en la tabla `ejecución`, que también contiene la información sobre el intento realizado y la prueba ejecutada.

Tipo	Descripción
Estructura	Las pruebas de estructura son las pruebas básicas. Consisten en comparar los métodos del alumno con la referencia proporcionada por el profesor.
Seguridad	Las pruebas de seguridad consisten en la ejecución de los métodos de la clase. Su función es detectar si los métodos lanzan excepciones, sin tener en cuenta el resultado.
Corrección	Las pruebas de corrección ejecutan los métodos públicos de la referencia proporcionada por el profesor, los del alumno y posteriormente comparan sus resultados.
Eficiencia	Las pruebas de eficiencia comparan la ejecución de los métodos públicos de la referencia proporcionada por el profesor y del alumno, en terminos de consumo de recursos, ya sean memoria o tiempo de ejecución.

Figura 3.4: Principales conjuntos de pruebas.

intento	prueba	resultado
1	1	0
1	2	0
2	1	10.0
2	2	10.0

En el registro podemos ver el resultado de realizar 2 intentos distintos sobre 2 pruebas. En el primer intento las pruebas no son superadas, y en el segundo sí lo son.

□

A partir de ahora, el módulo del motor encargado de la conexión lo llamamos **conector**.El **conector** lee de la base de datos la dirección de los códigos que queremos ejecutar. Estas direcciones son las rutas absolutas de la carpeta que representa la raíz del árbol de paquetes del código del i) alumno a evaluar y ii) del profesor.

Ejemplo 3.2 Continuando con el Ejemplo 3.1, podemos ver que en el registro **referencia** contiene la dirección del código del profesor.

id	codigo	creado
1	/home/www/web/files/referencias/1/	2012-01-15 02:52:11

En el registro **intento** encontramos la dirección del código a evaluar, así como el grupo de alumnos propietarios, la tarea de la que trata el intento, y guardaremos la calificación cuando se compute.

id	codigo	creado	calificacion	grupo	tarea
1	/home/www/web/files/intentos/1/	2012-01-15 02:53:21	*NULL*	1	1
2	/home/www/web/files/intentos/2/	2012-01-15 02:57:17	*NULL*	1	1

Además en el registro **prueba** encontramos los tipos de prueba que se puede aplicar y el tipo de resultado que generan.

id	nombre	tipo_resultado
1	Prueba de estructura	SI/NO
2	Prueba de seguridad	SI/NO

En este caso podemos observar dos tipos de pruebas posibles, que en este caso obtendrán como resultado éxito o fracaso.

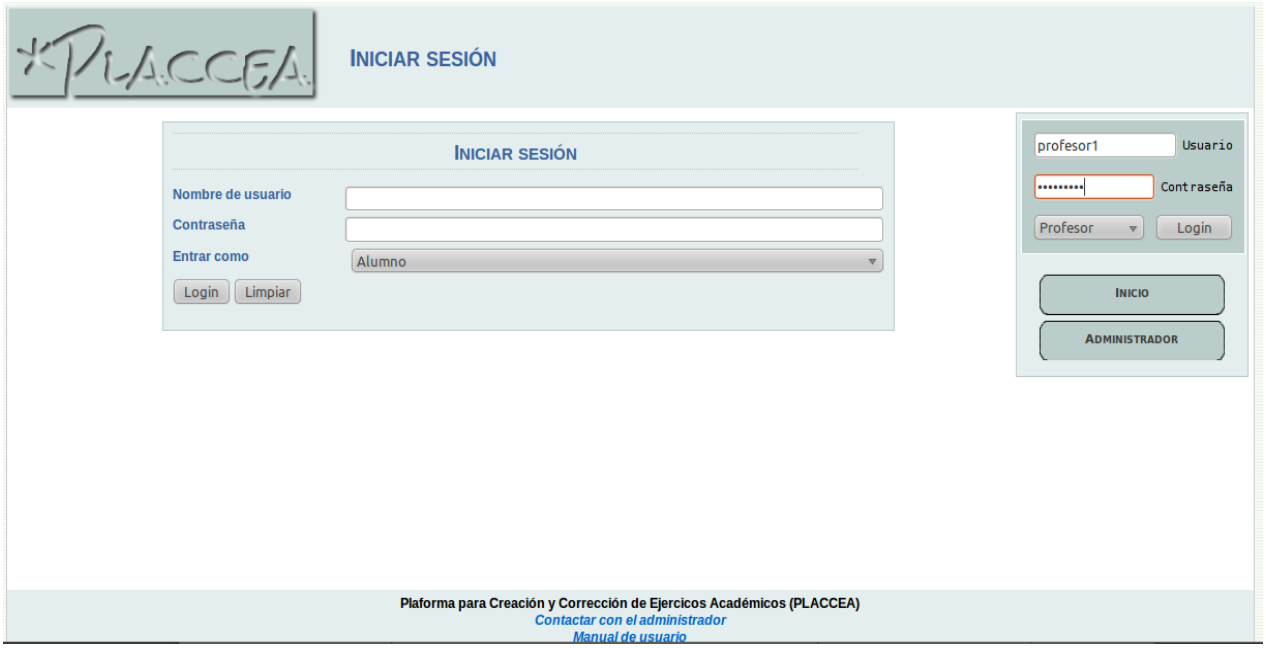
□

El **conector** además de los dos ficheros anteriores lee el tipo de prueba que queremos ejecutar. Una vez que el **conector** obtiene los datos, éste activa al **motor**. La figura 3.5 muestra las clases que se ven involucradas en el lanzamiento a ejecución de una prueba.

En la Figura 3.6 presentamos los comandos básicos que implementará el **motor**. A continuación vamos a ir detallando con más detalle cada uno de ellos.

Una vez que el **motor** tiene estos datos, se ejecuta la prueba utilizando el comando 1 de la Figura 3.6.

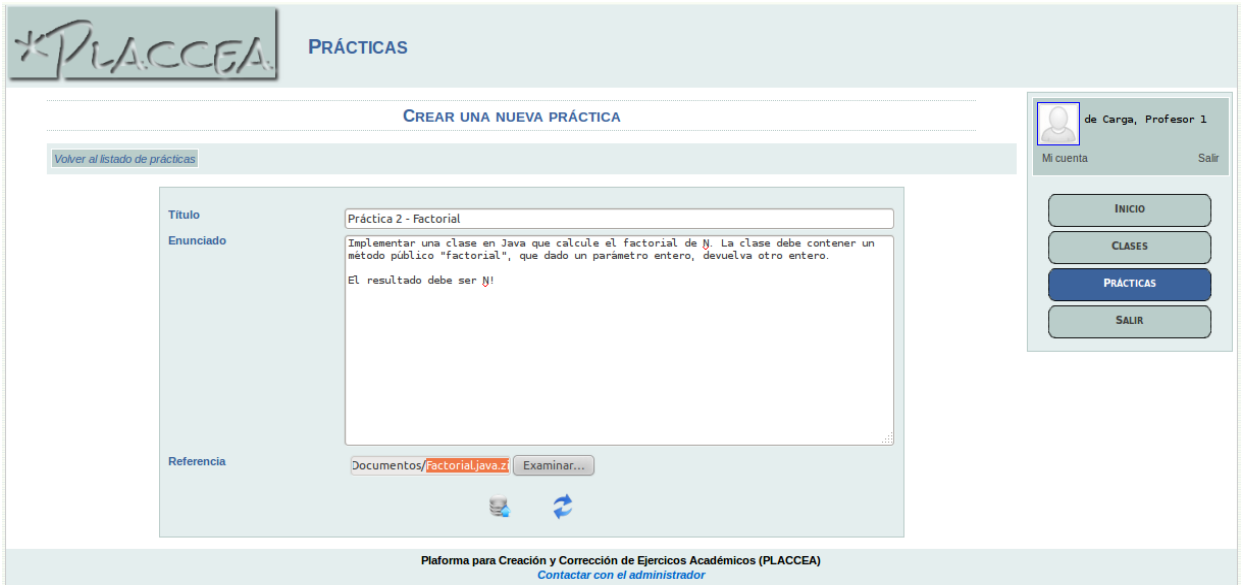
Ejemplo 3.3 Por ejemplo consideremos que el profesor `profesor1` quiere subir un nuevo ejercicio. Lo primero que tiene que hacer es entrar en la aplicación.



Crear un nuevo ejercicio.


INICIO
Bienvenid@ a la plataforma X~Project
El contenido de la plataforma está dividido en apartados accesibles mediante el menú principal situado en la parte derecha. A continuación se describe brevemente cada uno de estos apartados.
Clases: En este apartado se recoge todo lo relacionado con las clases en las que figura como docente. En este apartado encontrará:
Información de los grupos y alumnos de la clase
Crear una nueva tarea (con una práctica existente)
Información de las tareas de la clase
Resultados de los intentos de los alumnos
Diversa información estadística de la clase
Prácticas: En este apartado se recogen las prácticas disponibles en el sistema. Usted también puede crear una nueva práctica.
Ver información de las prácticas existentes
Descargar el código de referencia de una práctica
Crear una nueva práctica
Mi cuenta: En este apartado puede consultar la información relativa a su usuario en el sistema. En este apartado también puede:
Modificar su contraseña de acceso
Actualizar su imagen de usuario
Actualizar su dirección de correo electrónico
de Carga, Profesor 1
Mi cuenta
Salir
INICIO
CLASES
PRÁCTICAS
SALIR
Plaforma para Creación y Corrección de Ejercicos Académicos (PLACCEA)
Contactar con el administrador
Manual de usuario


PRÁCTICAS
PRÁCTICAS DISPONIBLES
Crear una nueva práctica
Título
Autor
Fecha de creación
Práctica 1 - Perros y pulgas
Profesor 4 de Carga
2012-04-14 22:06:06
de Carga, Profesor 1
Mi cuenta
Salir
INICIO
CLASES
PRÁCTICAS
SALIR
Plaforma para Creación y Corrección de Ejercicos Académicos (PLACCEA)
Contactar con el administrador
Manual de usuario



Merece la pena mencionar que las referencias se subirán como un archivo comprimido en formato `zip`. Esta decisión de diseño se toma porque es posible subir un conjunto de ficheros fuente para su correción, y se debe respetar aunque solo pretendamos subir un único fichero.

Una vez que ha subido la práctica, el profesor programa la tarea.


CLASES
CLASES EN LAS QUE ES PROFESOR
Asignatura
Curso
Clase
Tareas
Grupos
Alumnos
Asignatura 1
2011/2012
Clase 1
1 Tareas
5 Grupos
25 Alumnos
de Carga, Profesor 1
Mi cuenta
Salir
INICIO
CLASES
PRÁCTICAS
SALIR
Plaforma para Creación y Corrección de Ejercicos Académicos (PLACCEA)
Contactar con el administrador
Manual de usuario


CLASES
ASIGNATURA 1 | CLASE 1 - 2011/2012
Ver grupos en la clase
Ver tareas en la clase
Ver estadísticas de la clase
Crear una tarea para una práctica
de Carga, Profesor 1
Mi cuenta
Salir
INICIO
CLASES
PRÁCTICAS
SALIR
Plaforma para Creación y Corrección de Ejercicos Académicos (PLACCEA)
Contactar con el administrador
Manual de usuario

Cuando un alumno suba una solución a ese ejercicio, el se encola una petición de corrección y se invoca al **conector**. Posteriormente el **motor** ejecutará el comando `ejecuta(rutaProfesor,rutaAlumno,nombrePrueba)`.

□

Una vez que el motor tiene estos datos, recorreremos los dos directorios con fin de construir un objeto intermedio de clase `RuneableTrial` 3.5.Esta clase estructura la información como la necesitan las pruebas, es decir, pares (nombre de la clase | directorio raiz de la ruta de paquetes de la clase). Con la información estructurada, lo siguiente es compilarla (recordamos que los alumnos suben las fuentes). Para compilarlos utilizamos las clases `Runtime` y `Process`. Ahora para cada par, lanzaremos las pruebas con el comando 2 de la Figura 3.6.

Ejemplo 3.4 Continuando con el ejemplo anterior, cuando el **motor** ejecuta el comando 1, construye una lista de objetos `RuneableTrial`, y para cada uno de ellos, ejecuta el comando 2 de la Figura 3.6.

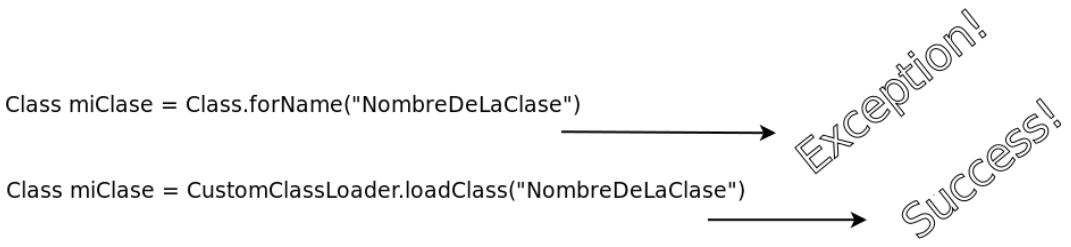
```
public static String ejecuta(String rutaP, String rutaA, String prueba) throws MotorException{
    /*
     * Construimos la lista de pares clase/directorio, que serán lo que se ejecute
     */
    double nota=0.0;
    try{
        ArrayList<RuneableTrial> lista_runeable = RuneableTrialCreator.create(rutaP,rutaA);
        compila(lista_runeable);
        Iterator<RuneableTrial> it=lista_runeable.iterator();
        boolean resultado=false;
        int intentos=0;
        int aciertos=0;
        while(it.hasNext()){
```

La descripción de los objetos `RuneableTrial` necesarios para ejecutar las pruebas, se encuentra en la figura 3.5.

```
while(it.hasNext()){
    RuneableTrial runeable=it.next();
    if(prueba.equals("Prueba de estructura")){
        resultado=RuneaTrial.runea(runeable.getClaseProfesor(),
                runeable.getClaseAlumno(), rutaP, rutaA, "PublicCompare");
    }else if(prueba.equals("Prueba de seguridad")){
        resultado=RuneaTrial.runea(runeable.getClaseProfesor(),
                runeable.getClaseAlumno(), rutaP, rutaA, "PublicExceptionTrial");
    }else if(prueba.equals("Prueba de correción")){
         resultado=RuneaTrial.runea(runeable.getClaseProfesor(),
                runeable.getClaseAlumno(), rutaP, rutaA, "PublicCorrection");
    }
    if(resultado){
        aciertos++;
    }
    intentos++;
```

□

Lo primero que tenemos que hacer para lanzar una prueba, es cargar las clases a probar en forma del objeto `Class<?>` correspondiente.Para hacerlo, primero intentaremos utilizar el procedimiento de reflexión `Class.forName(NombreClase)`.



Esto lógicamente no funcionará, pues las clases que queremos cargar están fuera del classpath del motor. Así que para cargarlas utilizaremos nuestro propio ClassLoader. 3.5.

En este caso nuestro CustomClassLoader es muy sencillo, solamente necesitamos extender URLClassLoader y de esta manera ya podemos cargar clases que se encuentren en cualquier parte del sistema de archivos. Con las clases ya cargadas, lanzamos finalmente las pruebas y recogemos su resultado.

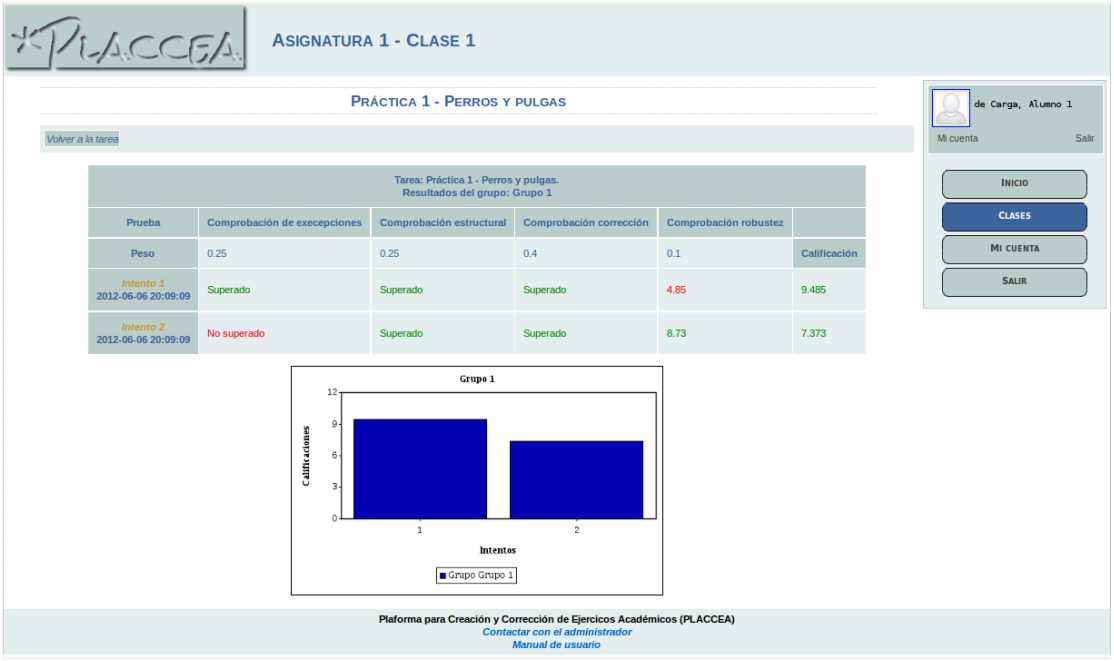
Ejemplo 3.5 Continuando con el ejemplo anterior, en el momento en que el **motor** ejecuta el comando 2, intenta cargar las clases necesarias.

```
try{
    claseP = Class.forName(claseProfesor);
    exitoP=true;
}catch(ClassNotFoundException e){}

if(!exitoP){
    try{
        loader.add(urlProfesor);
        claseP=loader.loadClass(claseProfesor);
    }catch(ClassNotFoundException e){
        throw new MotorException("No se ha podido cargar la clase referencia.");
    }
}
```

Una vez que hemos cargado las clases necesarias, ejecutamos la prueba correspondiente.

```
Trial compare;
if(tipo.equals(TipoDePrueba.PublicCompare.toString())){
    compare = new PublicCompare(claseP, claseA);
}else if(tipo.equals(TipoDePrueba.PublicCorrection.toString())){
    compare = new PublicCorrection(claseP, claseA);
}else{
    compare = new PublicExceptionTrial(claseA);
}

boolean resultado=compare.trial();
```

Posteriormente, tanto el alumno como el profesor, podrán ver las calificaciones.



□

El resultado de las pruebas será el tanto por ciento de éxito. Una prueba no se considera superada si su resultado no es `1.0`. Con los resultados de las pruebas componemos la nota, que será el número de clases que superan la prueba respecto del número total.

Para terminar, el motor devuelve la nota y el **conector** se encarga de guardarla en la base de datos. A continuación, presentamos el framework que implementa el motor, tanto en su parte **conector** como en su parte de ejecución.

El resto del capítulo se estructurará de la siguiente manera. En la Sección 3.1 se presentará el Framework del motor JAVA. En la Sección 3.2 se presentará el Generador de Estadísticas.

3.1. Framework

Desde el principio nuestra intención ha sido que tanto el tipo, como la calidad de las pruebas, fuera extensible. De esta forma, en la elaboración del motor de pruebas, decidimos componer un framework. La principal tarea del framework es dar soporte a la definición de nuevas pruebas. Incluímos utilidades para interacción con base de datos, además de otras clases importantes en la ejecución de las pruebas. Por último, incluye ejemplos de uso para los distintos tipos de pruebas.

Como complemento, incluímos en el framework la funcionalidad necesaria para ejecutar las pruebas sin necesidad de la plataforma, es decir, localmente. La base de datos tampoco es imprescindible. Dado la naturaleza del servidor donde se implementa la plataforma web, que se ejecuta sobre una distribución de GNU/Linux, el framework está pensado para trabajar también en estos entornos, de forma que el resultado es más homogéneo.

Nuestro framework trabaja usando reflexión en Java. Para ello utilizamos API Reflection de Java, incluído en el paquete java.lang.reflect. En la sección 3.1.1explicamos en que consiste la reflexión.

Ahora presentamos algunas de los componentes del framework. En la Sección 3.1.2 se presentará el **generador**. En la Sección 3.1.3 se presentará la Clase Trial.

3.1.1. Reflexión en Java

La reflexión es una carácteristica de la programación en Java. Permite a un programa examinar carácteristicas de si mismo en ejecución (tener una visión introspectiva de si mismo) y manipular propiedades internas del programa. Por ejemplo es posible navegar por todos los componentes de una clase y mostrarlos. La habilidad de examinar una clase Java desde si misma puede no parecer muy útil, pero en otros lenguajes de programación ni siquiera existe esa característica. Por ejemplo, en C++ lo máximo a lo que podríamos aspirar sería un mecanismo de reflexión pura basada en puntero-a-componentes.

Por lo tanto, usando reflexión podemos manipular las clases con gran libertad. Permite examinar una clase (y extraer toda su información), crear instancias y saltarnos todas las restricciones de acceso de forma dinámica. Pese a toda la potencia de la reflexión, nosotros nos centraremos en acceder a una clase, examinarla y ejecutar sus métodos. Todo esto se realiza de forma genérica. En la figura 3.7 mostramos la clase Información, que obtiene toda la información de una clase, obtenida de forma dinámica mediante reflexión.

Por otra parte no todo en la reflexión son ventajas. Su uso no es nada sencillo, algo que no pasa mucho con Java. Además, algunas de las optimizaciones que se realizan sobre el bytecode no se pueden ejecutar cuando usamos reflexión, porque la reflexión involucra tipos que se resuelven de forma dinámica, de forma que hay una pérdida de rendimiento considerable. Y eso no es todo, como usando reflexión podemos saltarnos los permisos de acceso, así que puede tener efectos laterales que pueden hacer que el código deje de funcionar o se pierda la portabilidad.

La ayuda para entender la reflexión la obtenemos de la documentación de Oracle [10].

A continuación mostramos algunos aspectos interesantes de la reflexión, enfocados como

curiosidades.

Constructor por defecto en Java

A través de este API, hemos aprendido que algunas historias que nos habían vendido sobre la programación orientada a objetos (como que todas las clases tienen una constructora sin argumentos por defecto) no son ciertas en Java. Un hecho fácil de desmentir, pues usando reflexión podemos recoger el conjunto de constructores de una clase y usarlos como mejor nos parezca.

Métodos públicos

En el contexto del uso de la reflexión, al referirse a métodos públicos, se refiere a todos los que se encuentran en la clase, no necesariamente a todos los que hayamos declarado. Esto es una distinción importante en nuestro contexto, porque si no tenemos cuidado podemos encontrarnos probando el método `hashCode()` o el método `equals(Object)`, que no son interesantes para nuestro propósito (evaluar el trabajo del alumno según una referencia). Nosotros por el contrario, al hablar de métodos públicos, nos referiremos a los métodos que se hayan declarado y además sean públicos.

3.1.2. Generador

Para que las pruebas sean lo más potentes posible, utilizaremos valores aleatorios en la construcción de los objetos. De esta manera intentamos contemplar el mayor abanico de posibilidades para la ejecución de pruebas. Usando reflexión construimos nuestro generador de objetos. En este caso la reflexión es una herramienta muy potente, pues con ella podemos examinar los constructores de una clase, y de entre ellos, elegir uno de forma aleatoria. Hay que mencionar que cada tipo de objeto (clase compuesta, array, tipo enumerado) se instancia de forma distinta con reflexión. Pese a todo, la reflexión nos ayuda a definir un Generador lo más aleatorio y completo posible.

El ciclo de generación de instancias es el siguiente:

1. Para recibir una instancia de la clase A, utilizaremos `Generador.getNewObject(Clase.class)`.
2. Lo primero que hace Generador, es comprobar si la clase es primitiva con `isPrimitive(Clase)`. En caso de que sea alguno de los tipos primitivos, devolvemos `getNewPrimitive()`, porque las instancias de objetos primitivos de generan de manera directa apoyandonos en la clase Random.

3. En caso de que la clase no sea primitiva, extraemos su conjunto de constructores mediante reflexión. Guardaremos en un HashMap la clase y sus constructores en pares clave-valor con el fin de convservarlos para futuros accesos.

4. De los constructores recogidos, seleccionaremos uno de forma aleatoria. Para el constructor seleccionado, extramos el conjunto de tipos de sus parámetros para su invocación.

5. Los parámetros para invocar al constructor, deben instanciarse. Para cada atributo, tendremos que distinguir si es un array, un tipo enumerado, o ninguno de los anteriores, pues cada uno se instancia de forma distinta.

   a) Si es un array, primero instanciamos el array de la determinada longitud, y posteriormente lo rellenaremos como objetos normales. La longitud del array será como máximo `MAX-LENGTH`. Este parámetro es un atributo de clase que nos ayuda a no generar arrays absurdamente grandes.

   b) Si es un tipo enumerado, leemos sus valores posible con `getEnumConstants()` y elegimos uno al azar.

   c) Para los objetos normales, les invocaremos por recursión. Esta es una decisión peligrosa, porque muchas clases de propósito similar (como String y StringBuffer) pueden construirse las unas a partir de las otras, creando ciclos. En este punto encontramos un dilema entre el compromiso con la aleatoriedad y la necesidad de escapar de estras trampas recursivas.

6. Finalmente, utilizaremos el constructor seleccionado y sus parametros previamente instanciados, para crear la instancia con el método `newInstance(args[])` de la clase Constructor del API de reflexión.

Ejemplo 3.6 Supongamos que tenemos la clase A definida de la siguiente manera:

Clase A
A ()
A (int)
A (String)

Nuestro generador aleatorio de objetos se comportaría de la siguiente forma:

1. Supongamos que se elige el constructor `A ()`. En este caso, la constructora es vacía, por lo que su parámetro es un array de objects con 0 elementos. Esto es importante. Si no lo hicieramos de esta manera es muy posible que recibamos una `InstantiationException`.

2. Supongamos que se elige el constructor `A (int)`. En este caso, construiremos el argumento de tipo int por recursión. Claro está, lo primero que hace la llamada recursiva es detectar que es de tipo primitivo, y devolver `Generador.getNewPrimitive()`.

3. Supongamos que se elige el constructor `A (String)`. En este caso, trataremos a la clase String como no-primitiva e intentaremos construir una instancia de manera recursiva.

□

Como el contexto de este proyecto es dar soporte a asignaturas de programación de los primeros cursos, cuyas prácticas son relativamente simples, no necesitamos toda la potencia de Generador, así que en la práctica utilizaremos el **GeneradorSimple**.

El **GeneradorSimple** tiene un dominio más concreto que Generador. Las clases que vamos a probar y que se les encarga componer a los alumnos de los primeros cursos, terminan por utilizar tipos básicos y String. GeneradorSimple construirá objetos que en su arbol de llamadas a constructoras, tengan tipos básicos y String en sus hojas. De esta forma, las diferencias básicas entre Generador y GeneradorSimple son:

1. Rango de los tpos básicos. En la clase Generador, los rangos son muy amplios, tan amplios como los tipos (int, double, float) nos permitan representar. Para el contexto de la aplicación, resulta más cómodo movernos por rangos más "normales". Este rango será un atributo de la clase, por lo que podremos ajustarlo según necesitemos.

2. Consideramos String como una clase básica y la construiremos de forma directa.

Ejemplo 3.7 Supongamos que queremos construir una instancia de la clase del ejemplo anterior 3.6 utilizando **GeneradorSimple**.

1. Para el caso en que se elija el constructor sin argumentos, se comporta igual que **Generador**. 1

2. Para el caso en que se elija el constructor `A (int)`, se invocará `getNewSimpleInteger()` para construir el parámetro del constructor, que construye un valor entero positivo menor o igual que el parámetro de clase `MAX-INT`.

3. Para el caso en que se elija el constructor `A (String)`, devolverá `getNewString()` para construir el argumento. En este método, se trata a String como un tipo básico y se devuelve un array de carácteres de longitud máxima `MAX-LENGTH`.

□

Todo esto se traduce en introducir "impurezas" en GeneradorSimple, reduciendo el uso de reflexión, con el fin de que resulte más adecuado y sencillo que Generador.

3.1.3. Clase Trial

Para definir las pruebas de forma homogénea, montaremos una estructura que parte de la clase abstracta Trial. Esta decisión se cimenta en una idea clara que tuvimos desde el primer momento: al hacer las pruebas polimórficas, se facilita el diseño, el uso y la claridad. En un principio pensamos en una interfaz, pues podía resultar interesante tener más libertad a la hora de crear pruebas. Finalmente optamos por una clase abstracta, que aporta todo lo que necesitamos.

De esta clase heredaran todas las pruebas, y se define de esta forma.

`boolean trial` Esta función realiza la prueba. Es decir, devuelve el resultado de la prueba, guarda el resultado dentro del parámetro, calcula el porcentaje, y en caso de encontrarse un error, lo registra y almacena la causa del error.

`String getMensajeError` Esta función devuelve la causa del error en caso de haberse producido. Conviene comentar que el mensaje se inicializa a un texto vacío.

`boolean hayError` Esta función devuelve el atributo error. Como puede ser razonable, el atributo error siempre se inicializa a falso.

`double getPorcentaje` Esta función devuelve el porcentaje de acierto. Es útil en los casos en que la prueba no se supera, pues el porcentaje es orientativo respecto a la futura corrección de los posibles errores que contenga la/s clase/s.

En la Figura 3.9 se muestra el código de la clase Trial.

A continuación presentamos diferentes casos de estudio. En particular en el primero de ellos estudiamos la implementación de una prueba de estructura. 3.1.3 En el segundo estudiamos la implementación de una prueba de seguridad. 3.1.3 En el tercero estudiamos la implementación de una prueba de corrección. 3.1.3 Finalmente en el cuarto, estudiamos la implementación de una prueba de eficiencia. 3.1.3

Caso de estudio 1: Estructura

En este caso de estudio, examinaremos una implementación de prueba de estructura, la clase `PublicCompare`. Esta implementación extiende la clase Trial que hemos definido en la Sección 3.1.3, y se construye con la información que contienen la clase que define el profesor como referencia de código correcto, y la clase que el alumno quiere probar.

Para hacer esto, utilizaremos la clase intermedia `InformacionPublica`, que extiende la clase `Informacion` 3.7, mencionada anteriormente en la sección 3.1.1 como ejemplo de la información que podemos extraer de una clase mediante reflexión.

En la figura 3.10 mostramos el diagrama UML de la clase `PublicCompare`.

El objetivo de esta prueba es comprobar si la estructura de ambos códigos es similar, es decir, si los métodos públicos de la clase del profesor, se encuentran en la del alumno. Basta con que coincidan nombre, parámetros, tipo devuelto y excepciones lanzables (con esto nos referimos a similar). En caso de que algún método público del profesor no se encuentre en la clase del alumno, la prueba fallará. También fallará si se encuentran atributos públicos en la clase del alumno.

En concreto para gestionar la información de los métodos utilizamos la clase `Metodo`. Esta clase simplemente almacena datos del método, como nombre, tipo de sus parámetros, tipo devuelto, execepciones lanzables, y pueden contener una instancia concreta de parametros para su invocación y el resultado de ejecutarlo con dichos parámetros, aunque estas características nos interesarán más tarde.

De esta forma, la clase `InformacionPublica` construye una lista con un atributo `Metodo` por cada método público declarado. Esto se hace así debido a las carácterísticas de la reflexión. 3.1.1

Dicho esto, para utilizar la prueba lo único que necesitamos usar son las siguientes instrucciones:

```
1    Class<?> claseP = ejemplo.profesor.Ejemplo.class;
2    Class<?> claseA = ejemplo.alumno.Ejemplo.class;
3    Trial compare = new PublicCompare(claseP, claseA);
4    boolean resultado=compare.trial();
```

Al ejecutar `compare.trial()`, lo primero que hace la implementación es buscar atributos públicos en la clase del alumno. Si ninguno de sus atributos es público, recogemos los métodos públicos del profesor en una colección de objetos `Metodo` (así es como se almacenan en `InformacionPubica`) y de igual manera recogemos los métodos públicos de la

clase del alumno. Para cada método de la colección de métodos del profesor, buscamos su equivalente en la colección de métodos del alumno. Esto lo hacemos sólamente con `metodosAlumno.contains(metodoProfesor)`. Esto funciona porque la clase `Metodo` redefine el método `equals(Object)` para que cumpla los requisitos que hemos mencionado al comienzo de la descripción de la prueba, y precisamente, si hicieramos esto mismo con objetos de tipo `Method` en lugar de con nuestra clase intermedia, no funcionaría.

El resultado de la prueba será, por tanto, la proporción de métodos definidos por el profesor encontrados entre los métodos públicos del alumno. Esta prueba es eliminatoria, por tanto sólo consideraríamos un éxito encontrar el 100 % de los métodos.

La razón por la que consideramos esta prueba como eliminatoria, es que otras (como la prueba de corrección que veremos a continuación) comparan los métodos del alumno con los del profesor; lo cual no tiene sentido si los métodos no son parejos.

Ejemplo 3.8 Continuando con el ejemplo anterior, cuando el alumno solicita la ejecución de una prueba de estructura, lo hace a través de esta implementación.

```
Trial compare;
if(tipo.equals(TipoDePrueba.PublicCompare.toString())){
    compare = new PublicCompare(claseP, claseA);
}else if(tipo.equals(TipoDePrueba.PublicCorrection.toString())){
    compare = new PublicCorrection(claseP, claseA);
}else{
    compare = new PublicExceptionTrial(claseA);
}

boolean resultado=compare.trial();
```

Primero comprobamos los atributos de la clase del alumno.

```
 Field[] atributosAlumno = _infoAlumno.getAtributosPublicos();
 if(atributosAlumno.length>0){
     _error=true;
     _mensajeError="Existen atributos publicos";
```

Después comparamos los métodos y calculamos el resultado.

```
int cuantosSon=metodosProfesor.size();
int encontrados=0;
Iterator<Metodo> itProfesor = metodosProfesor.iterator();
while(itProfesor.hasNext()){
    Metodo metodoProfesor=itProfesor.next();
    if(metodosAlumno.contains(metodoProfesor)){
        encontrados++;
    }
}
//Si hemos encontrado todos los metodos, hemos pasado la prueba.
this._porcentaje=encontrados/cuantosSon;
if(this._porcentaje==1.0){
    _resultado=true;
}
```

□

Caso de estudio 2: Seguridad

En este caso de estudio, examinaremos una implementación de prueba de seguridad, la clase `PublicExceptionTrial`. Esta implementación extiende la clase Trial que hemos definido en la Sección 3.1.3, y se construye con la clase proporcionada por el alumno.

Esta prueba consiste en la ejecución de métodos, por lo que necesitamos una instancia de la clase que el alumno está probando, para lo que utilizamos la clase `GeneradorSimple`, presentada en la sección 3.1.2.

En la figura 3.11 mostramos un diagrama UML de la clase `PublicExceptionTrial`.

El objetivo es ejecutar todos los métodos públicos y comprobar si se han lanzado excepciones. Si esto ocurre, y la excepción lanzada no se encuentra en la declaración del método como una excepción lanzable, la prueba fallará. Como el abanico de posibilidades es muy grande, esta acción se ejecuta un número de iteraciones que por defecto es 10. Aumentar este valor, vuelve la prueba más exaustiva y precisa, pero reduce su rendimiento.

A diferencia de el caso de estudio anterior, esta clase no necesita utilizar la clase `Metodo` ni la clase `InformacionPublica`. La razón es que, simplemente, invocaremos a todos los métodos declarados en la clase, esperando el lanzamiento de alguna excepción. En caso de que se lanze alguna excepción, se captura y se busca dentro de las excepciones incluídas en la cabecera del método. Si no se lanza una excepción, o si se lanza pero estaba declarada dentro de la cabecera, lo consideraremos un exito. De no ser así, lo consideraremos un fallo.

Dicho esto, para utilizar la prueba lo único que necesitamos usar son las siguientes instrucciones:

```
1    Class<?> claseA = ejemplo.alumno.Ejemplo.class;
2    Trial compare = new PublicExceptionTrial(claseA);
3    boolean resultado=compare.trial();
```

En el atributo porcentaje guardaremos el número de aciertos entre el número de métodos a probar. También vamos a considerar esta prueba como eliminatoria, pues en la prueba de corrección de más tarde, queremos estar lo más seguros posible de que la ejecución de la clase no produce excepciones no capturadas.

Ejemplo 3.9 Continuando con el ejemplo anterior, cuando el alumno solicita la ejecución de una prueba de seguridad, lo hace a través de esta implementación.

```
Trial compare;
if(tipo.equals(TipoDePrueba.PublicCompare.toString())){
    compare = new PublicCompare(claseP, claseA);
}else if(tipo.equals(TipoDePrueba.PublicCorrection.toString())){
    compare = new PublicCorrection(claseP, claseA);
}else{
    compare = new PublicExceptionTrial(claseA);
}
```

Primero generamos una instancia de la clase.

```
Object obj=null;
try{
    obj = GeneradorSimple.getNewObject(_clase);
}catch (GeneradorException e){
    error=true;
}
```

Después, para cada método instanciamos sus argumentos.

```
while (i<parametros.length && !fin){
    try{
        parametros[i]=GeneradorSimple.getNewObject(clasesParametros[i]);
    }catch(GeneradorException e){
        _mensajeError+="Los parámetros del método "+m.getName()
                +"no se han podido construir.";
        fin=true;
    }finally{
        i++;
    }
```

Finalmente invocamos al método.

```
try{
    if(parametros.length>0){
        m.invoke(obj, parametros);
    }else{
        m.invoke(obj);
    }
    /*
     * Si todo ha ido bien, nos contamos el punto.
     */
    exitoMetodos++;
```

Repetiremos esto tantas veces como iteraciones hayamos fijado.

□

Caso de estudio 3: Corrección

En este caso de estudio, examinaremos una implementación de prueba de corrección, la clase `PublicCorrection`. Esta implementación extiende la clase Trial que hemos definido en la Sección 3.1.3, y se construye con la clase proporcionada por el profesor como referencia de código correcto y la proporcionada por el alumno.

Esta prueba consiste en ejecutar todos los métodos públicos del profesor, guardar su resultado y los argumentos de la llamada, ejecutar después los del alumno con los mismos argumentos y comparar su resultado. Para hacer esto último, volvemos a utilizar la clase `Metodo`, que como ya hemos comentado antes, nos permite guardar la instancia de los argumentos utilizados en la llamada, y el resultado de su invocación. Igual que en le caso de estudio anterior, estamos ejecutando métodos, por lo que recurriremos otra vez a la clase `GeneradorSimple`, presentada en la sección 3.1.2.

En la figura 3.12 mostramos un diagrama UML de la clase `PublicCorrection`.

Podemos ejecutar la prueba con las siguientes instrucciones:

```
1    Class<?> claseA = ejemplo.alumno.Ejemplo.class;
2    Class<?> claseP = ejemplo.profesor.Ejemplo.class;
3    Trial compare = new PublicExceptionTrial(claseP,claseA);
4    boolean resultado=compare.trial();
```

El resultado de la prueba será el número de métodos que coinciden en su resultado, tanto en el código del profesor, como en el del alumno; entre el total de métodos existentes en la clase proporcionada por el profesor. Tenemos el 100 % de acierto, lo consideraremos un éxito.

En el caso de que intentemos ejecutar esta prueba sin antes ejecutar con éxito la prueba de estructura, puede ocurrir que los métodos no se correspondan entre sí, y en ese caso, el resultado de la prueba será un porcentaje muy bajo, o incluso '0'.

Es importante remarcar las limitaciones de esta prueba. Para que funcione correctamente esta prueba debe aplicarse a clases que tenga unos metodos tales que, su resultado sólamente depende de el valor de los argumentos. Esto por si sólo nos hará dejar de considerar todos los métodos void, pues no podemos comparar su resultado. De este modo, la prueba esquiva los métodos que no generan resultados útiles. En concreto, no se aplica a los ya considerados métodos `void`, ni a `getters`, `setters` o métodos `toString`.

Caso de estudio 4: Eficiencia

En este caso de estudio, examinaremos una implementación de prueba de eficiencia, la clase `PublicPerformance`. Esta implementación extiende la clase Trial que hemos definido en la Sección 3.1.3, y se construye con la clase proporcionada por el alumno, y la referencia proporcionada por el profesor.

Esta prueba consiste en ejecutar todos los métodos públicos del profesor y medir el tiempo que tardan en ejecutarse. Para comparar posteriormente con los métodos del alumno,

guardaremos el tiempo de ejecución y además los argumentos con los que se invocó a cada método. Después ejecutaremos los del alumno con los mismos argumentos y comparar su tiempo de ejecución. Para hacer esto último, volvemos a utilizar la clase `Metodo`, que además de lo que hemos comentado en casos de estudio anteriores, nos permite también guardar el tiemo de ejecución. Igual que en casos de estudio anteriores, estamos ejecutando métodos, por lo que recurriremos otra vez a la clase `GeneradorSimple`, presentada en la sección 3.1.2.

En la figura 3.13 mostramos un diagrama UML de la clase `PublicPerformance`. Podemos observar que este diagrama es muy similar al del caso de estudio anterior.

Como en casos de estudio anteriores, podemos ejecutar la prueba con las siguientes instrucciones:

```
1    Class<?> claseA = ejemplo.alumno.Ejemplo.class;
2    Class<?> claseP = ejemplo.profesor.Ejemplo.class;
3    Trial compare = new PublicPerformance(claseP,claseA);
4    boolean resultado=compare.trial();
```

El resultado de la prueba será el porcentaje de métodos que se han ejecutado en un tiempo similar o menor a los métodos proporcionados por el profesor. En concreto, y como mera decisión de diseño, se considera un éxito si el tiempo de ejecución del alumno es menor o igual que T*1.1 donde T es el tiempo medido en la ejecución de los métodos del profesor.

Para ejecutar esta prueba, antes debería pasarse la prueba de corrección, pues si los resultados no son correctos, no importará que sean más eficientes en tiempo. Sin embargo, dadas las limitaciones de la prueba de corrección, no siempre estaremos en condiciones de exigir pasarla antes de ejecutar la prueba de eficiencia.

3.2. Generador de Estadísticas

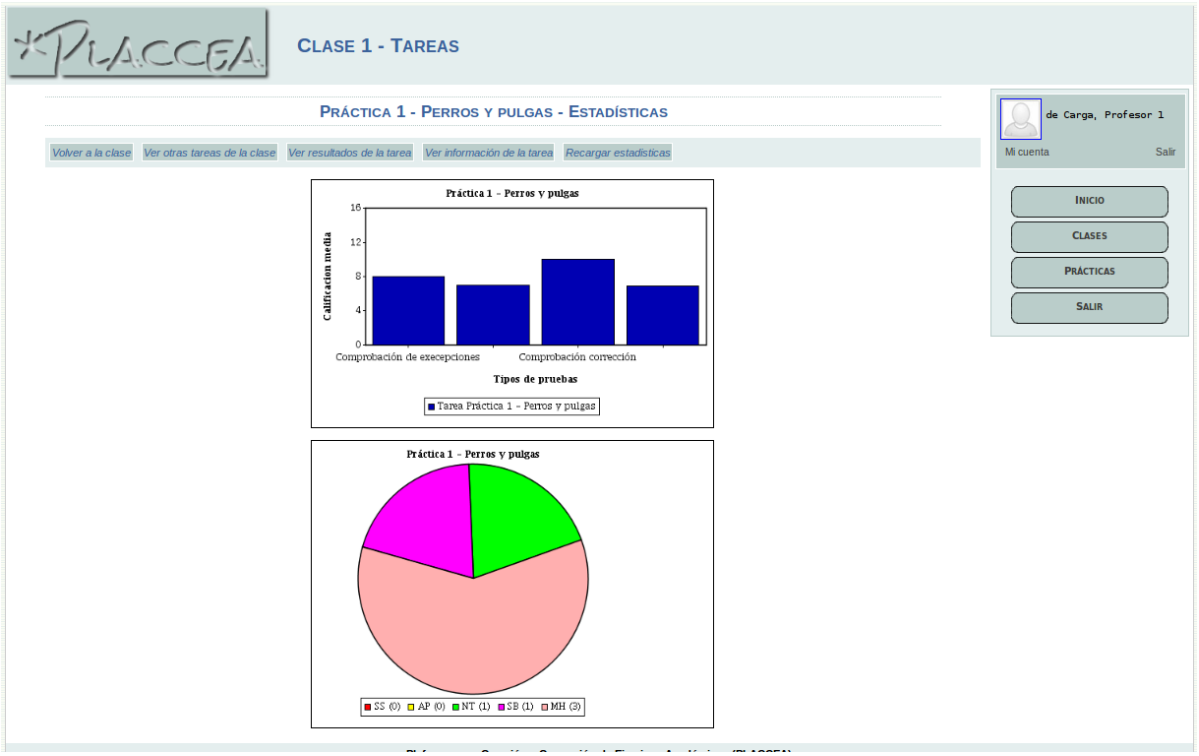
Aportamos la funcionalidad de generar estadísticas. Estas estadísticas se generan a partir de los resultados de las pruebas de los alumnos.

Para generar los gráficos de las estadísticas usamos la herramienta **jCharts**, de código libre e implementada en Java. Dicha herramienta se ha integrado en el proyecto, y permite la generación de distintos tipos de gráficos adaptando sus características y parametrizando los datos a representar.

Respecto del punto de vista del diseño, y para una mayor estructuración, se ha intregado mediante dos capas separadas. Una, que podemos llamar inferior, comunica con el paquete **jCharts** directamente, y aprovecha sus funciones para generar gráficos definidos y adaptados específicamente para nuestro sistema. La otra capa, que llamaremos superior se comunica con esta, usando los sevicios que ofrece la inferior para crear las instancias concretas de los gráficos. Esta capa es la que recibe los parametros que identifican que gráfico se debe crear.

Se invoca desde la web, pero es un paquete independiente que ofrece una interfaz cerrada para una mayor independencia entre las distintas partes del sistema. Este diseño permite una fácil implementación de nuevos tipos de gráficos, así como la readaptación de los ya existentes en caso de cambiar los requisitos.

Ejemplo 3.10 Siguiendo con los ejemplos anteriores, el profesor podrá consultar los resultados de las tareas:



□

Respecto del punto de vista de la implementación, la capa superior recibe como argumentos el tipo de gráfico y el identificador de la base de datos para esa instancia. Posteriormente recupera los datos de la base de datos, los trata, e invoca al método apropiado de la capa inferior. Esta capa se encarga de estructurar y preparar los gráficos según módelos predefinidos para nuestro sistema.

De esta forma obtiene un gráfico adecuado para la visualización de los resultados y actividades de los alumnos, partiendo de los datos registrados previamente en la base de datos. Además, posibilita la actualización al acceder a los gráficos, ofreciendo siempre la última información disponible.

motorjava.runtime

RuneableTrial

La clase RuneableTrial contiene toda la información necesaria para la ejecución de pruebas. En este nivel, las rutas constituyen la ruta absoluta a un fichero que implementa una clase en Java, y los nombre de las clases se representan de forma canónica, incluyento todo el arbol de paquetes. La clase se estructura de esta forma:

1. Ruta de la clase referencia proporcionada por el profesor.
2. Nombre de la clase referencia.
3. Ruta de la clase a del alumno.
4. Nombre de la clase del alumno.

RuneableTrialCreator

La clase RuneableTrialCreator implementa un único método estático, que recibe la ruta raíz del código fuente del alumno, y de la referencia proporcionada por el profesor, y devuelve una lista de elementos RuneableTrial.

CustomClassLoader

Como los códigos que pretendemos probar están fuera del classpath del **motor**, implementamos nuestro propio ClassLoader para cargar las clases por sus URL.

RuneaTrial

La clase RuneaTrial implementa un único método estático que recibe los argumentos contenidos en un objeto de la clase RuneableTrial y además el nombre de la Clase de prueba que queremos ejecutar. Antes de ejecutar la prueba, utiliza la clase CustomClassLoader para cargar las clases, y en caso de éxito, ejecutará la prueba y devolverá el resultado.

Figura 3.5: Clases involucradas en la ejecución de una prueba. Se encuentran dentro del paquete motorjava.runtime.

1. `ejecuta(rutaProfesor,rutaAlumno,nombrePrueba)`
2. `runea(claseProfesor,claseAlumno,urlProfesor,urlAlumno,tipoPrueba)`

Figura 3.6: Lista de Instrucciones empleadas en el motor.

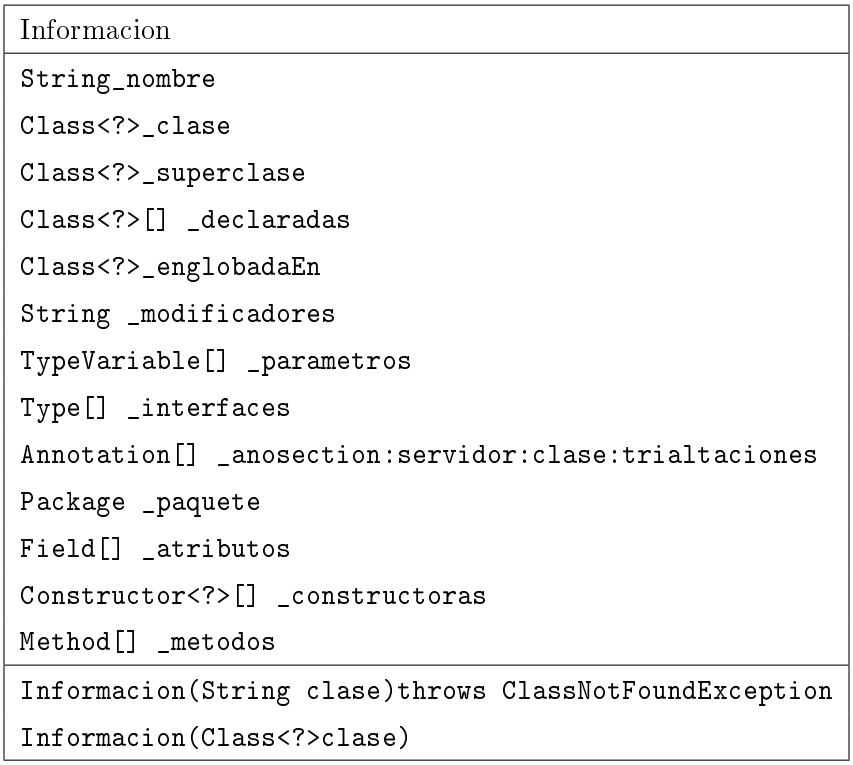


Figura 3.7: Diagrama UML clase `Informacion`.

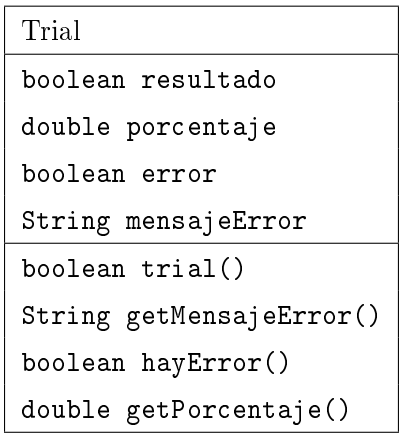


Figura 3.8: Diagrama UML clase `Trial`.

```java
package motorjava.trial;

/**
 * Define un conjunto de pruebas que se caracterizan por tener un resultado
     binario.
 * @author ssii_projectx
 */
public abstract class Trial {

    protected boolean _resultado;
    protected double _porcentaje;
    protected boolean _error;
    protected String _mensajeError;
    /**
     * Devuelve el resultado booleano de ejecutar la prueba.
     * @return
     */
    public abstract boolean trial();
    public abstract String getMensajeError();
    public abstract boolean hayError();
    public abstract double getPorcentaje();
}
```

Figura 3.9: Código de la clase Trial.

PublicCompare
`InformacionPublica infoProfesor` `InformacionPublica infoAlumno`

`PublicCompare(String profesor, String alumno)`
`PublicCompare(Class<?>profesor,Class<?>alumno)`
`boolean trial()`
`String getMensajeError()`
`boolean hayError()`
`double getPorcentaje()`

Figura 3.10: Diagrama UML clase `PublicCompare`.

PublicExceptionTrial
`Class<?>clase` `static int iteraciones`

`PublicExceptionTrial(Class<?>claseAlumno)`
`boolean trial()`
`String getMensajeError()`
`boolean hayError()`
`double getPorcentaje()`
`int getIteraciones()`
`void setIteraciones(int)`

Figura 3.11: Diagrama UML clase `PublicExceptionTrial`.

PublicCorrection
`Class<?>claseProfesor` `Class<?>claseAlumno`

`PublicCorrection(Class<?>profesor, Class<?>alumno)`
`boolean trial()`
`String getMensajeError()`
`boolean hayError()`
`double getPorcentaje()`

Figura 3.12: Diagrama UML clase `PublicCorrection`.

PublicPerformance
`Class<?>claseProfesor` `Class<?>claseAlumno`

`PublicPerformance(Class<?>profesor, Class<?>alumno)`
`boolean trial()`
`String getMensajeError()`
`boolean hayError()`
`double getPorcentaje()`

Figura 3.13: Diagrama UML clase `PublicPerformance`.

CAPÍTULO 4

Plataforma PLACCEA

En este capítulo presentaremos la plataforma PLACCEA, que constituye el principal punto de interacción de los usuarios con el sistema. Tanto alumnos como profesores tendrán acceso a esta plataforma on-line en donde podrán realizar la entrega de ejercicios, consultar los resultados de los ejercicios ya entregados, asignar nuevas tareas... y otras funcionalidades que permiten, en definitiva, facilitar la entrega de los ejercicios y acercar los resultados a los usuarios, que son dos de los principales objetivos de nuestro sistema informático. La plataforma PLACCEA está ubicada en un servidor de la Facultad de Informática de la UCM y es accesible desde: `https://projectx.fdi.ucm.es/web/`
La estructura de este capítulo es la siguiente:

- 4.1 **Análisis de requerimientos.** En esta sección exponemos los requisitos funcionales de nuestro sistema informático elaborados a partir de la propuesta inicial del proyecto, de las indicaciones de nuestro director de proyecto y de nuestras propias ideas.

- 4.2 **Especificación.** En esta sección realizamos una especificación formal de nuestro sistema, para ello definimos los roles de usuarios y los casos de uso de nuestro sistema.

- 4.3 **Arquitectura.** En esta sección se presenta la arquitectura cliente-servidor de la plataforma y su integración con el resto de componentes de nuestro sistema informático.
- 4.4 **Diseño.** Aquí se tratan los aspectos más relevantes del diseño del sistema.
- 4.5 **Implementación.** En esta sección se detallan los aspectos más relevantes de la implementación de la plataforma.
- 4.6 **Documentación.** Elaboraremos una serie de manuales que contribuyen a la documentación de la aplicación, con el fin de facilitar su uso y su evolución.
- 4.7 **Mantenimiento.** Para facilitar el mantenimiento implementaremos algunas herramientas de diagnóstico y resolución de posibles problemas que puedan aparecer y que provocarían la inestabilidad del sistema.

4.1. Análisis de requerimientos

El punto de partida del análisis de requisistos de nuestro sistema informático es la propuesta inicial del proyecto que hemos presentado. En primer lugar expondremos esta propuesta inicial, para posteriormente analizar los requisistos funcionales que aparecen de una manera informal en este documento, y ampliaremos este análisis con las indicaciones y especificaciones de nuestro director de proyecto.

4.1.1. Propuesta inicial del proyecto

Queremos desarrollar una aplicación cliente-servidor como apoyo a la asignatura "Tecnología de la Programación" de los nuevos grados que pueda ayudar a la corrección y evaluación de los alumnos. Dicha aplicación constituirá una plataforma on-line a la que alumnos y profesores de la asignatura podrán acceder para realizar distintas actividades relacionadas con los ejercicios prácticos de la asignatura. A continuación se detallan los servicios que la plataforma ofrecerá a alumnos y profesores:

Los alumnos podrán consultar en la plataforma los enunciados de los distintos ejercicios prácticos de la asignatura y podrán entregar dichos ejercicios resueltos (subiendo los archivos desde la aplicación cliente). Cada ejercicio consistirá en el desarrollo de un programa (en lenguajes como Java, C, C++ o Phyton) que realice una tarea especificada en el propio enunciado del ejercicio, junto con la especificación de los formatos de los datos de entrada y salida del programa. También podrán probar sus prácticas, según la configuración del profesor, previamente a la corrección de las mismas.

Los profesores tendrán también acceso a la plataforma. Entre los servicios que la plataforma proporcionará a los profesores se encuentra una herramienta de autoría de ejercicios, mediante la cual un profesor pueda definir el enunciado de un nuevo ejercicio y todo lo necesario para poder ser añadido y evaluado. También podrán consultar los ejercicios entregados por sus alumnos así como los resultados y estadísticas obtenidos a partir de la realización de una serie de pruebas de evaluación sobre cada ejercicio entregado por cada alumno.

Como se ha mencionado en el párrafo anterior, los ejercicios entregados por los alumnos serán sometidos a una serie de pruebas de evaluación (concretas para cada ejercicio y especificadas en el momento de su inclusión en la plataforma) de manera automática tras su entrega. Los resultados de estas pruebas serán almacenados en una base de datos para su posterior consulta por parte de los profesores y para la elaboración de estadísticas que resulten útiles para la evaluación de los ejercicios, o para que los alumnos vean sus avances y puedan situarse respecto al resto de la clase.

Entre los tipos de pruebas que se podrán realizar sobre los ejercicios entregados aparecen: pruebas de corrección (la salida del programa coíncide con la esperada para unos datos de entrada concretos), pruebas de rendimiento (memoria utilizada, tamaño de la pila de llamadas para algoritmos recursivos, número de instrucciones ejecutadas, tiempo de ejecución, ...), pruebas sintácticas (comparar diferencias entre los códigos de diferentes alumnos, detectando posibles copias, número de variables utilizadas, ...).

Para la realización de estas pruebas, se implementará un entorno de ejecución virtual, que evite los problemas generados por un código erróneo o mal intencionado.

En el momento de redactar esta propuesta queda sin zanjar la discusión sobre la tecnología a utilizar en la elaboración de la aplicación cliente. Las dos opciones que se barajan en este momento son: a) un cliente web o b) un cliente Java. Ambas opciones presentan características particulares cuyas ventajas e inconvenientes merecen ser estudiados con mayor detenimiento.

También se contempla la posibilidad de integrar esta plataforma con el LMS Moodle utilizado en el Campus Virtual de la UCM. Dicha integración consistiría en la implementación de un bloque de Moodle que constituya una interfaz de la plataforma adaptada a dicho LMS.

4.1.2. Análisis de la propuesta

En la propuesta expuesta en el apartado anterior podemos identificar claramente que habrá dos principales tipos de usuarios en la plataforma: alumnos y profesores. A continuación se detallan los requisitos funcionales que hemos podido extraer a partir de la propuesta

inicial.

- En la plataforma hay dos tipos de usuarios: alumnos y profesores.
- Debe existir un control de acceso a la plataforma que impida el acceso público.
- Los profesores deben poder acceder a la información y funcionalidades relacionadas con las clases de las asignaturas de las que es profesor, y con los alumnos de los que es profesor. No deben ver la información de otros alumnos que no sean los suyos, ni de otras clases de las que no sea profesor.
- Los alumnos deben poder acceder únicamente a la información y funcionalidades relacionadas con las clases en las que está matriculado.
- Un ejercicio consiste en el desarrollo de un programa que realice una tarea especificada en el propio enunciado del ejercicio.
- La entrega de la resolución de un ejercicio consiste en la entrega del código del programa que resuelve (o no) la tarea especificada en el enunciado.
- Los profesores deben poder crear ejercicios desde la plataforma. Deben poder elaborar los enunciados de los ejercicios prácticos. En estos enunciados incluirán información acerca del formato de la entrada y salida del programa (en caso de existir).
- Los profesores deben poder configurar las pruebas que se deben realizar sobre las resoluciones de los ejercicios para su evaluación.
- Los profesores deben poder consultar los programas entregados por sus alumnos. También deben tener acceso a los resultados de las pruebas realizadas sobre estos programas.
- Los alumnos deben poder consultar los enunciados de los ejercicios que han sido asignados a las clases en las que es alumno.
- Los alumnos deben poder enviar a través de la plataforma sus resoluciones de los ejercicios.
- Los alumnos deben tener acceso a los resultados de las pruebas realizadas sobre sus programas.
- Al crear un ejercicio, el profesor que lo cree debe adjuntar su resolución correcta del ejercicio, que se tomará como referencia para corregir los de los alumnos.

- Cuando un alumno entregue una resolución de un ejercicio, de manera automática el motor de pruebas realiza sobre éste las pruebas que se hayan indicado para el ejercicio, tras lo cual guarda los resultados en una base de datos.
- La plataforma debe presentar estos resultados de una manera legible a los usuarios que, según lo descrito anteriormente, tengan acceso a estos.
- La plataforma debe poder presentar estadísticas relevantes para profesores y/o alumnos que sean elaboradas a partir de estos resultados.

4.1.3. Ampliación de la propuesta

Una vez expuesto el análisis de la propuesta inicial de nuestro sistema informático, procedemos a exponer el resto de requisitos funcionales que hemos ido elaborando a partir de las conversaciones que hemos mantenido con nuestro director de proyecto, y otros elaborados por nosotros mismos para concretar detalles que no aparecen en la propuesta inicial.

- Los alumnos se organizarán en grupos de trabajo, dado que es la forma más habitual de trabajar.
- Los alumnos de un mismo grupo tendrán acceso a la misma información, a excepción de su información personal.
- El profesor podrá fijar un plazo de entrega para los ejercicios resueltos de los alumnos a la hora de crear una tarea.
- El profesor podrá decidir las pruebas que se realizarán para cada tarea. Para ello, podrá escoger de entre un conjunto de pruebas disponibles en la plataforma.
- La calificación de un ejercicio entregado por un grupo se compondrá como una media ponderada tomando los resultados de las distintas pruebas como valores y unos pesos que el profesor podrá asignar al crear la tarea.
- Además de profesores y alumnos, también habrá un usuario administrador encargado de gestionar aquellos aspectos ajenos a profesores y alumnos, como la creación de las clases o la asignación de los alumnos y profesores a sus respectivas clases.
- El usuario administrador además podrá realizar otras tareas relacionadas con el mantenimiento de la plataforma, como comprobar el estado actual del sistema.

4.2. Especificación

El objetivo de este apartado es exponer de una manera formal la especificación de la plataforma PLACCEA. Para ello, haremos uso de diversos diagramas UML [1] que hemos elaborado a partir del análisis funcional del apartado anterior.

En primer lugar, presentaremos los diagramas de casos de uso de nuestra plataforma. El objetivo de estos diagramas es poder transmitir al lector de una manera sencilla las funcionalidades que estarán disponibles para cada tipo de usuario. Este apartado se complementa con los diagramas de actividades que aparecen en el capítulo 3.

4.2.1. Casos de uso

A partir del análisis llevado a cabo en los apartados anteriores hemos elaborado los diagramas de casos de uso que aparecen a continuación. En estos diagramas se indica de una manera clara y sencilla la funcionalidad de la aplicación y la funcionalidad disponible para cada uno de los actores que participan. El objetivo de estos diagramas es ilustrar de una manera visual la funcionalidad de la aplicación y las acciones disponibles para cada uno de los tres actores en el sistema: alumno, profesor y administrador.

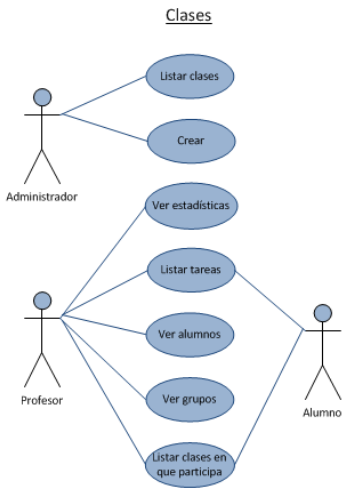


Figura 4.1: Casos de uso - clases

[1] UML(Unified Modeling Language) es un lenguaje de modelado de sistemas propuesto por la OMG (Object Management Group). Más información en `http://www.omg.org/`)

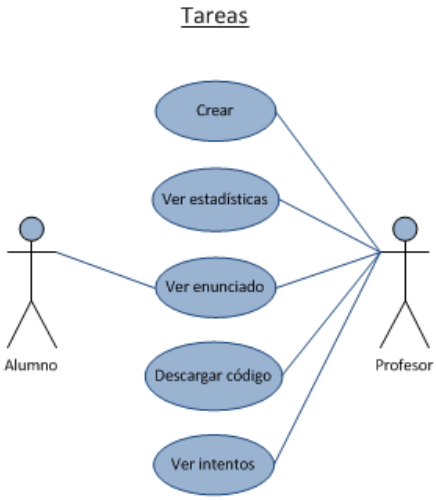


Figura 4.2: Casos de uso - tareas

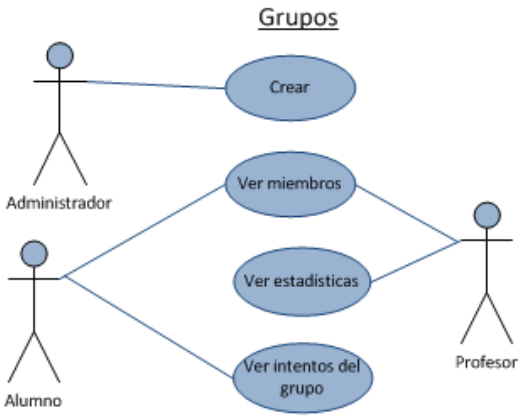


Figura 4.3: Casos de uso - grupos

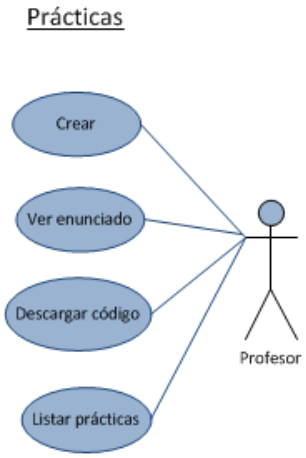


Figura 4.4: Casos de uso - prácticas

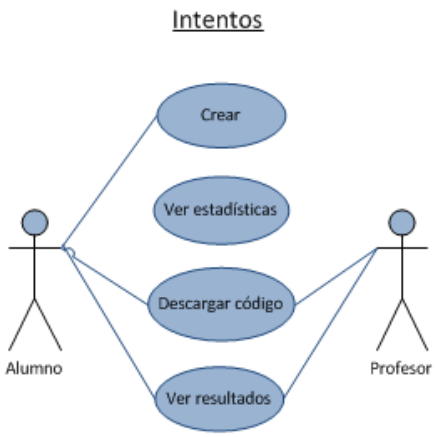


Figura 4.5: Casos de uso - intentos

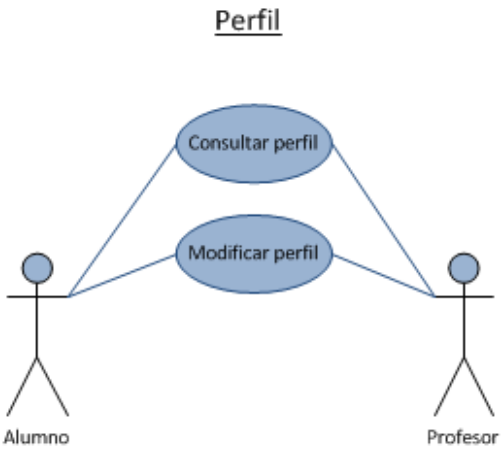


Figura 4.6: Casos de uso - perfil de usuario

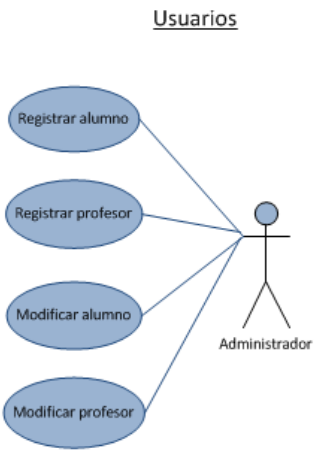


Figura 4.7: Casos de uso - usuarios

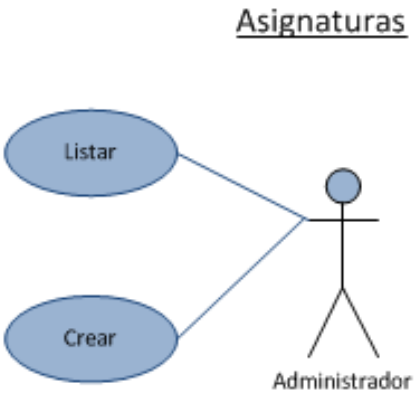


Figura 4.8: Casos de uso - asignaturas

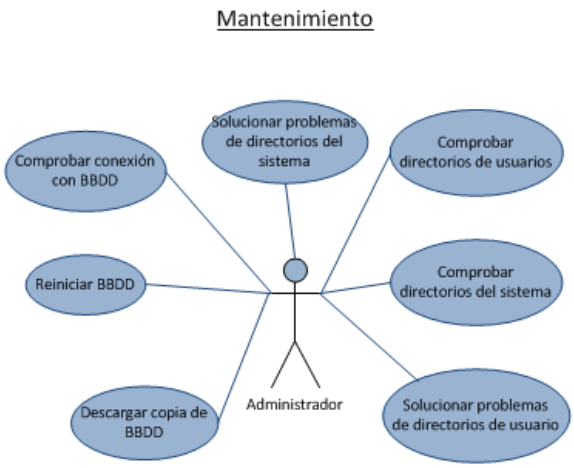


Figura 4.9: Casos de uso - mantenimiento

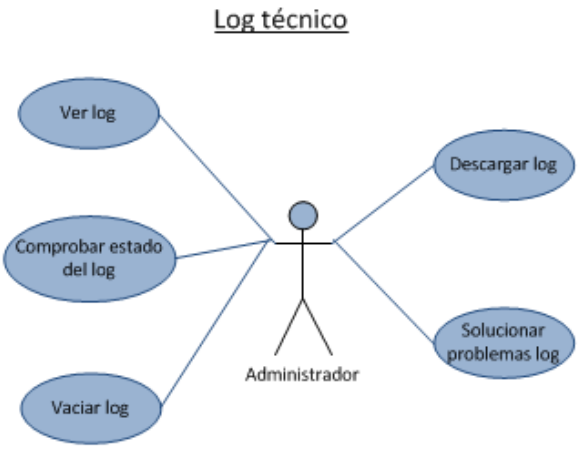


Figura 4.10: Casos de uso - log técnico

4.3. Arquitectura

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, que son llamados servidores, y los consumidores de estos servicios, que son llamados clientes. El flujo de trabajo consiste básicamente en que el cliente envía solicitudes al servidor, el cual las procesa y produce una respuesta que es devuelta al cliente.

Esta es la arquitectura que seguirá nuestro sistema informático. Para tomar esta decisión nos hemos basado en los siguientes aspectos:

- Uno de nuestros objetivos es producir resultados estadísticos relevantes a partir de los resultados de las pruebas realizadas sobre los programas de los alumnos. Un resultado estadístico relevante para un profesor puede ser algo tan simple como, por ejemplo, conocer el número de alumnos que han superado una prueba para una determinada práctica. Para poder elaborar este tipo de resultados necesitamos disponer de los resultados de las prácticas de todos los alumnos de esa clase y para poder realizar esta tarea con eficiencia, necesitamos poder acceder a ellos de una manera centralizada. Si situamos estos datos en el servidor (por ejemplo, en una base de datos) podremos acceder a todos ellos con facilidad y de manera eficiente.

- El motor JAVA, que será el componente del sistema encargado de realizar la evaluación de los programas, es uno de los componentes más susceptibles de ser actualizado y modificado a lo largo de la vida del sistema. Si optamos por implementar este componente de manera distribuida, es decir, que se ejecute en el computador del usuario, estaremos indirectamente asumiendo la responsabilidad de distribuir tanto la versión inicial de este programa como sus futuras actualizaciones. En cambio si optamos por situar este componente en el servidor no será necesario distribuirlo a los clientes. Además, dado que como se ha expuesto en el apartado anterior, existe la necesidad de disponer de los resultados producidos por este componente de manera centralizada, al situarlo en el servidor estaremos favoreciendo la comunicación entre este componente y el almacén de datos.

- El profesor necesita disponer del código de los programas entregados por los alumnos para su evaluación. Otro de nuestros objetivos consiste en facilitar esta entrega. Situar los programas entregados por los alumnos en un servidor es una medida que facilita esta tarea, al poder disponer de todos ellos de una manera centralizada. Tomar esta decisión

nos obliga a asumir la responsabilidad de garantizar el correcto almacenamiento de estos programas así como de restringir el acceso a éstos.

4.4. Diseño

Con la información presentada en las secciones anteriores de este mismo capítulo se puede proceder al diseño de la aplicación web que constituirá el punto de interacción con los usuarios del sistema. A lo largo de la presente sección expondremos los aspectos más relevantes del diseño de la aplicación, que definen gran parte de la implementación que se presenta en el siguiente capítulo.

La estructura de este apartado es la siguiente:

- 4.4.1 **Modelo Vista Controlador**. En este apartado presentamos el patrón de diseño Modelo-Vista-Controlador, sus principales ventajas y la motivación que nos ha llevado a escogerlo como el patrón que guíe la arquitectura de la aplicación web.
- 4.4.2 **Diseño de la base de datos**. En este apartado exponemos el diseño de la base de datos que constituirá el modelo de datos de nuestra aplicación utilizando diagramas de Entidad-Relación.
- 4.4.3 **Diseño de la interfaz gráfica de usuario**. Finalizamos esta sección con unos bocetos del diseño de la interfaz gráfica de usuario que servirán como orientación para su implementación.

4.4.1. Modelo Vista Controlador

A la hora de diseñar e implementar una aplicación de cierta embergadura como la que aquí presentamos, es altamente recomendable por no decir indispensable aplicar patrones de diseño. El objetivo de su utilización se puede resumir en la cita de Christopher Alexander:

> *"Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez"* `Christopher Alexander - The Timeless Way of Building`

En nuestra aplicación web aplicaremos el patrón de diseño estructural Modelo-Vista-Controlador (MVC). El primer motivo de esta decisión es que todos los miembros del equipo

estamos familiarizados con él. Además, es uno de los patrones más aplicados en las aplicaciones web actuales y existen multitud de frameworks disponibles que ahorrarán mucho trabajo en la implementación. El concepto de Framework se explica con más detalle en el apartado 4.5.2 titulado CodeIgniter Framework. A continuación explicaremos en qué consiste MVC.

Este patrón, que fue descrito por primera vez en 1979 por Trygve Reenskaug[2], consiste resumidamente en dividir la aplicación en tres componentes principales:

- **Modelo:** trabaja principalmente con los datos de la aplicación. En aplicaciones web, lo más común dado su funcionamiento, es persistir los datos de la aplicación sobre una base de datos. El modelo es el componente que realiza peticiones a la base de datos y habitualmente también se encarga de convertirlos en estructuras de datos que faciliten su representación en la vista. En la subsección 4.4.2 expondremos con detalle el diseño de la base de datos.

- **Vista:** este componente es el encargado de implementar la interfaz gráfica de usuario. En aplicaciones web, este componente es el encargado de generar el código Html que será enviado al explorador del cliente. En la subsección 4.4.3 entraremos en más detalle sobre el diseño de la interfaz gráfica de usuario.

- **Controlador:** se encarga de atender las peticiones del usuario. Cuando se recibe una petición, el controlador realiza las llamadas necesarias al modelo para leer o modificar los datos de la aplicación y posteriormente realiza las llamadas oportunas a la vista para generar la respuesta al usuario.

En la figura 4.11 se describe el flujo de trabajo de una aplicación web con arquitectura MVC. En él se aprecia cómo el punto de interacción entre el explorador web del cliente y la aplicación en el servidor es el controlador. También se describe cómo el controlador realiza peticiones parametrizadas al modelo, el cual responde con datos que son enviados, también por el controlador, a la vista. Por último, la vista genera el contenido de la GUI y, a través del controlador, es enviado al cliente. En la subsección 4.5.2 titulada CodeIgniter Framework veremos cómo este esquema puede adaptarse a sistemas más complejos que requieren una división en componentes más concreta.

[2] Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC) por Trygve Reenskaug

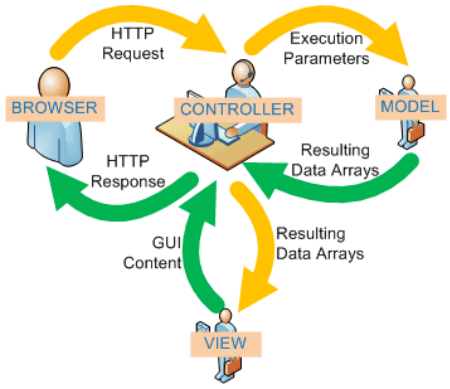


Figura 4.11: Esquema del flujo de trabajo del patrón MVC aplicado a aplicaciones web

4.4.2. Diseño de la base de datos

Como se ha comentado en el apartado anterior, lo más común en las aplicaciones web actuales para persistir los datos de la aplicación es hacer uso de un Sistema Gestor de Bases de Datos (SGBD). Ampliando lo explicado al comienzo del capítulo 3 acerca de cómo un servidor se encarga de despertar los procesos que atienden las peticiones, en una aplicación web cada petición HTTP que llega al servidor se traduce en una ejecución de la aplicación, que finaliza una vez producida la respuesta HTTP. Esto provoca que los datos almacenados en la memoria de la que el programador hace uso son liberados tras atender cada petición, lo cual crea una necesidad evidente de persistir los datos en un sistema de almacenamiento como un disco duro.

El uso de un SGBD proporciona una serie de ventajas que lo convierten en el sistema más utilizado para cubrir esta necesidad. Por una parte, la tarea de asegurar un buen uso del espacio lógico disponible y de garantizar la integridad de los datos es delegada al SGBD librando al programador de la aplicación de esta responsabilidad. Por otra parte, facilita el acceso a la información gracias al uso de los denominados lenguajes de consulta de bases de datos[3] y al repertorio de operaciones disponibles.

En la figura 4.12 se representa el diseño del modelo de datos de la aplicación haciendo uso de los diagramas Entidad - Relación. De esta manera podemos definir el modelo de datos sin concretar la tecnología que se va a utilizar para su implementación. Para una mejor comprensión de estos diagramas, hemos omitido en el diagrama principal los atributos de las entidades que no forman parte de su clave primaria. En la figura 4.13 se describen los

[3]Los lenguajes de consulta de bases de datos permiten describir operaciones de alta, baja, modificación y consulta sobre una base de datos relativamente complejas en un lenguaje de alto nivel.

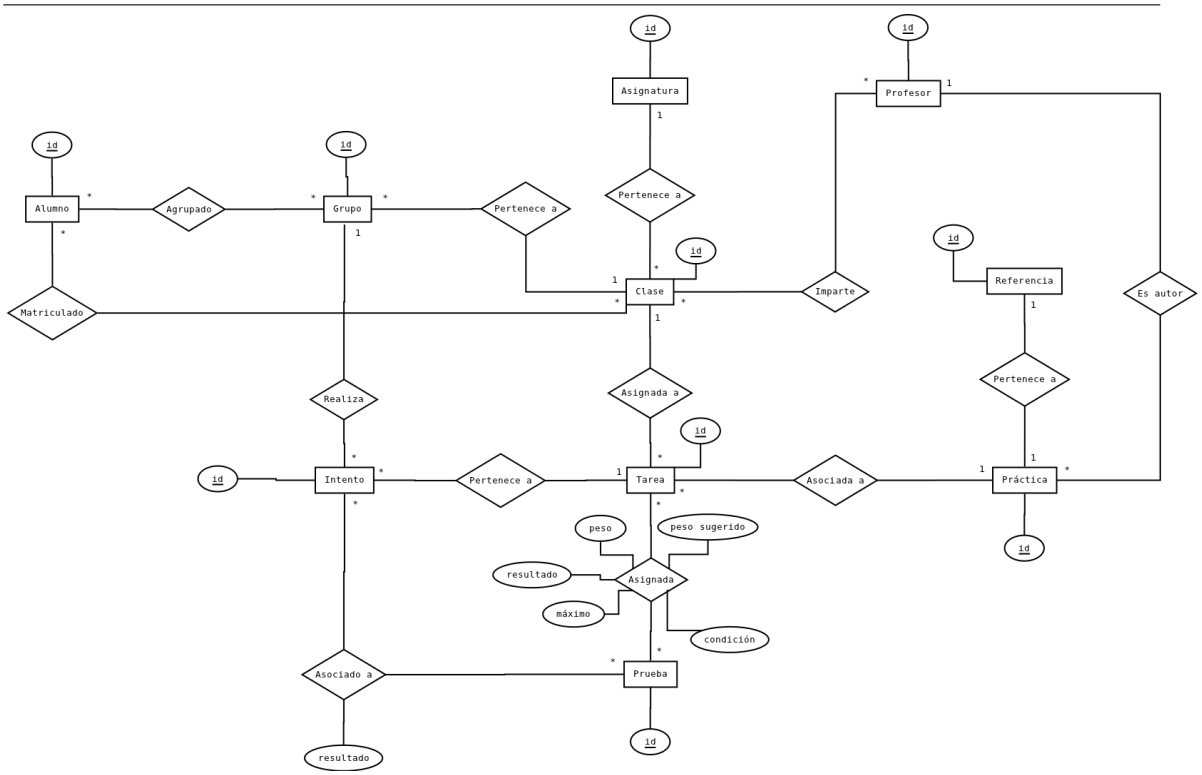


Figura 4.12: Diagrama Entidad - Relación del modelo de datos de la aplicación

atributos que componen cada entidad.

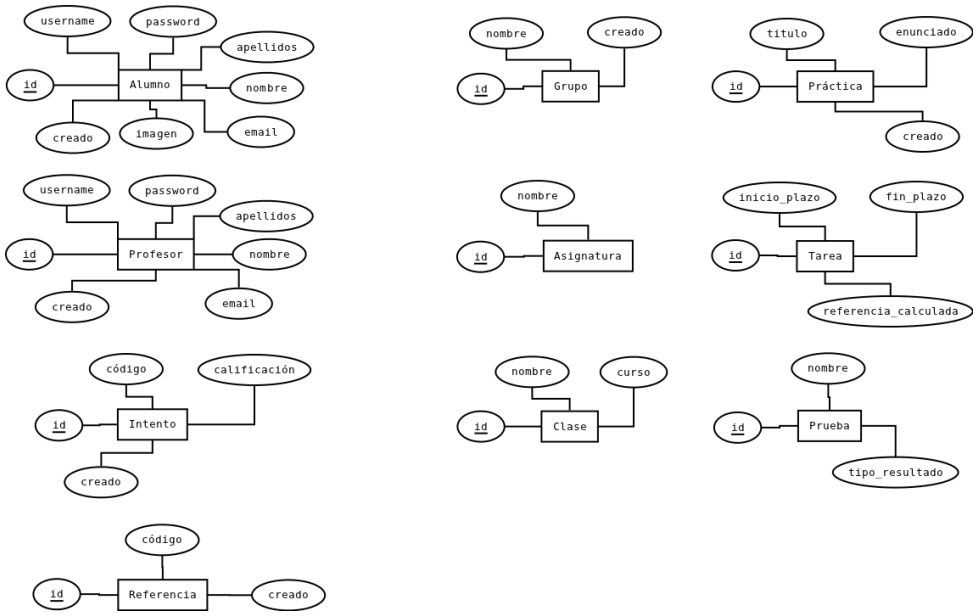


Figura 4.13: Diagrama Entidad - Relación en el que se detallan los atributos de las entidades

4.4.3. Diseño de la interfaz gráfica de usuario

Se denomina Interfaz Gráfica de Usuario (GUI) al punto de interacción de un usuario con un sistema informático mediante una presentación gráfica de la información y de los controles. Uno de los criterios de calidad de software que ha cobrado mayor importancia en los últimos años es la usabilidad, concepto definido por la ISO [4] de la siguiente manera:

> *"La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso"* ISO/IEC 9126

Dado que uno de nuestros objetivos es facilitar la entrega de los ejercicios por parte de los alumnos, así como facilitar el acceso a los resultados por parte de los profesores, diseñar una interfaz usable y accesible es un factor determinante para el logro de nuestros objetivos. A continuación describiremos el diseño general de la GUI de nuestro sistema haciendo uso de bocetos.

En la figura 4.14 se muestra el diseño general de la interfaz gráfica para un usuario autenticado en el sistema. Está basada en la estructura habitual de las páginas más comunes, con el fin de facilitar la navegación del usuario inexperto en la plataforma. A continuación

[4] Organización internacional para la Estandarización

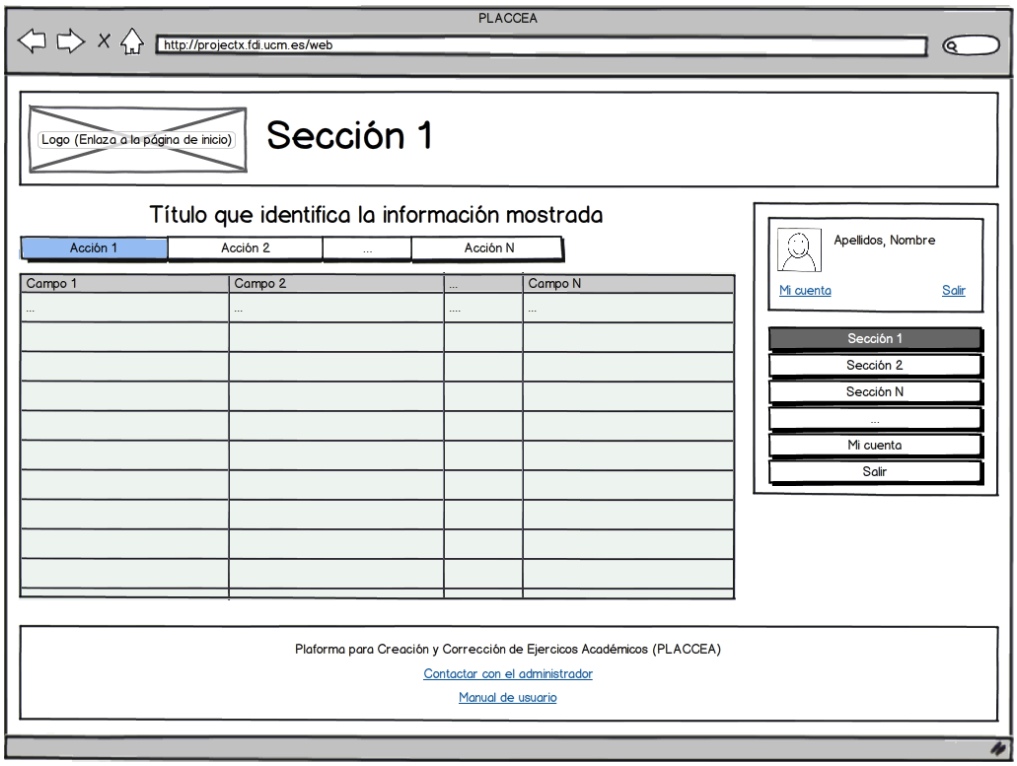


Figura 4.14: Boceto representativo del diseño general de la GUI

describiremos los componentes más relevantes:

- **Logo:** situado en la parte superior izquierda estará el logo de la plataforma en forma de imagen. Además estará enlazada de manera que al pulsar sobre él, el usuario se dirigirá a la página de inicio.
- **Menú principal:** en la parte derecha de la página se observa un menú que permite acceder a las distintas secciones. El contenido de este menú variará según el rol del usuario. Se puede apreciar que la sección en la que se encuentra actualmente el usuario aparece destacada y su nombre aparece en el título de la parte superior, a la derecha del logo de la plataforma.
- **Título descriptivo:** bajo el título principal de la página y el logo aparecerá un título que identificará el contenido que se está visualizando en este momento. Por ejemplo, si el usuario es un profesor que está viendo los resultados de un intento de un grupo, este título identificará al grupo y al intento concreto.
- **Información del usuario:** situado en la parte derecha de todas las páginas aparecerá

una caja con la información del usuario actual. Esta caja contendrá una imagen del usuario, su nombre completo, y dos enlaces que le permitan gestionar los datos de su cuenta y desautenticarse.

- **Menú de acciones:** aparece situado bajo el título descriptivo. El contenido de este menú variará en función del rol del usuario y de la página que esté visitando actualmente. A través de este menú el usuario accederá a otras páginas que permitan realizar acciones relacionadas con el contenido actual. Por ejemplo, si el usuario es un profesor que está viendo la información de una clase, aquí aparecerán acciones como "Crear una nueva tarea" o "Ver grupos de la clase".

- **Contenido/información:** en la parte central estará la información de la página actual. La mayoría de la información se mostrará en tablas con el fin de facilitar su visualización y homogeneizar el diseño. Con esta decisión también esperamos facilitar su implementación.

- **Pie de página:** en la parte inferior de todas las páginas aparecerá un pie de página en el que figurará el nombre completo de la plataforma, un enlace para ponerse en contacto con el administrador y un enlace al manual de usuario, del que hablaremos en la sección 4.6.1

4.5. Implementación

En este apartado expondremos el diseño y la implementación de la aplicación que hemos elaborado a partir de los requisitos y las especificaciones que se recogen en los apartados anteriores. El objetivo de esta sección es proporcionar al lector una visión clara y general de los aspectos más relevantes del diseño, así como introducir las tecnologías que se utilizarán para su implementación y justificar su elección.

La estructura de este apartado es la siguiente:

- 4.5.1 **Tecnologías y lenguajes de programación:** en este apartado especificaremos las tecnologías que se utilizarán en la implementación de la aplicación, los lenguajes de programación que emplearemos y los motivos que nos han llevado a tomar estas decisiones.

- 4.5.2 **CodeIgniter Framework:** en este apartado se introduce el framework sobre el cuál se construirá la aplicación y las algunas de las ventajas que su uso nos ofrece.

- 4.5.3 **Modelo de datos:** aquí se describe la implementación de la capa de Modelo.
- 4.5.4 **Vistas:** aquí trataremos la capa de Vista.
- 4.5.5 **Controladores:** en esta sección se describe la capa de Controlador.
- 4.5.6 **Librerías de acceso a datos:** forman parte de la capa de modelo, aquí se tratan en mayor profundidad.
- 4.5.7 **Integración con el resto del sistema:** cómo la aplicación web se integra con el resto de componentes.

4.5.1. Tecnologías y lenguajes de programación

En primer lugar comenzaremos definiendo las tecnologías y lenguajes de programación que vamos a utilizar para la implementación de la plataforma web, justificando las decisiones tomadas por el equipo.

Dado que se trata de una plataforma web, evidentemente utilizaremos el lenguaje **Html**, como ya se ha explicado en el apartado 4.4.1 titulado Modelo Vista Controlador. Más concretamente, utilizaremos la versión 4.01 Strict[5] recomendada por por la organización w3c [20]. La decisión de utilizar este estándar se apoya en que es reconocido por todos los navegadores web actuales.

También utilizaremos el lenguaje **Javascript** como lenguaje de código de cliente, con el único fin de modificar algunos comportamientos de los elementos Html de la interfaz gráfica del usuario y el lenguaje **Css**[6] para construír el diseño de la GUI especificado en el apartado 4.4.3, en concreto utilizaremos el estándar de css 2.1 recomendado también por la organización w3c [20]. De nuevo, esta decisión se debe a que esta versión es reconocida por todos los navegadores actuales.

Como lenguaje para la implementación del servidor web de la plataforma,utilizaremos **Php**. En primer lugar escogemos este lenguaje por ser libre [7], en segundo lugar porque varios miembros del equipo estamos familiarizados con él, y en tercer lugar por la cantidad de Frameworks disponibles publicados también bajo licencias de Software Libre. En la sección 4.5.2 definirimos el concepto de Framework.

[5] La especificación del estándar Html 4.01 se puede encontrar aquí: `http://www.w3.org/TR/REC-html40/`

[6] Css: Cascading Style Sheets o en español Hojas de Estilo en Cascada es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML.

[7] El lenguaje Php está publicado bajo licencia PhpLicense 3.01. Esta licencia puede encontrarse aquí: `http://www.php.net/license/3_01.txt`

El sistema gestor de base de datos que utilizaremos es **MySQL** [9]. En primer lugar escogemos esta tecnología por estar publicada bajo una licencia de código libre, además Php ofrece una muy buena integración con ella, y en tercer lugar porque el lenguaje de consulta utilizado por este sistema es **SQL**, lenguaje que conocemos todos los integrantes del grupo y con el que estamos familiarizados.

4.5.2. CodeIgniter Framework

Codeigniter [8] es un Framework publicado bajo licencia de código libre y escrito en el lenguaje php. Sigue la arquitectura Modelo-Vista-Controlador explicada en el apartado 4.4.1 y la amplía introduciendo otros componentes como controladores, librerías, ayudantes, seguridad... explicaremos algunos de estos conceptos a lo largo de esta sección.

Un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado.

En la figura 4.15 se puede ver el flujo de trabajo del framework. Sin entrar en detalles irrelevantes para el objetivo de este documento, describiremos brevemente lo que esta figura representa: todas las peticiones al servidor pasan por el archivo index.php, el cual inicializa la aplicación y realiza la carga de todos los componentes ésta. A continuación, el componente denominado "Routing" se encarga de procesar la url de la petición y la traduce en una llamada a un método de uno de los controladores. Seguidamente aparece un componente que implementa la capa de seguridad del sistema, en este componente se comprueba la autenticación del usuario.

A partir de aquí comienza el código que deberemos implementar, ya que los componentes anteriores forman parte del núcleo de CodeIgniter. Se realiza la llamada al método del controlador que se haya determinado con los argumentos adecuados, este controlador hará uso de los distintos componentes transversales que aparecen en la parte derecha de la figura, entre los que cabe destacar: Modelos, Librerías, Drivers y Helpers. Una vez el controlador ha finalizado su tarea, envía los datos oportunos a la vista adecuada, generando la página en Html que será enviada al explorador del cliente. Opcionalmente se puede habilitar un componente denominado "Caching" que forma parte del núcleo de CodeIgniter y que permite utilizar una caché de páginas generadas, con el fin de aumentar el rendimiento de la aplicación.

4.5.3. Modelo de datos

La capa de modelo de nuestra aplicación está constituída por un conjunto de clases que se pueden agrupar en dos conjuntos: aquellas que representan entidades y aquellas que presentan relaciones.

Para simplificar la implementación todas las entidades del modelo descrito en el apartado 4.4.2 en el que se describe el diseño de la base de datos tendrán como clave primaria un campo

[8] Página Oficial de CodeIgniter: `http://www.codeigniter.com`

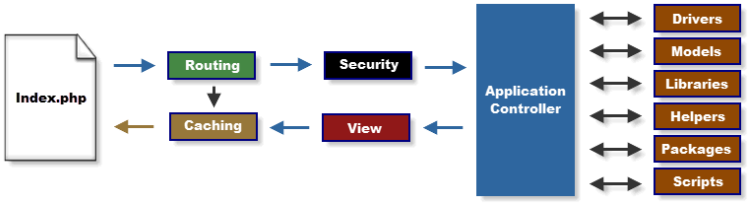


Figura 4.15: Flujo de trabajo de CodeIgniter

autoincremental denominado identificador o id. Al implementar las tablas de la base de datos de esta manera, es sencillo parametrizar las operaciones de consulta para todas las clases que representan una entidad. Esta agrupación queda reflejada en la figura 4.16 como la clase Modelo-base de la cual heredan el resto de clases. Las operaciones que se implementan en esta clase son las operaciones básicas de alta, baja, modificación y consulta.

Las clases que representan relaciones del modelo de datos siguen una agrupación similar, heredando todas de la clase Modelo-rel-base, la cual implementa los métodos comunes a todas estas clases. En este caso, los métodos de consulta son más variados ya que cada relación tiene unas necesidades concretas. En la figura 4.17 se puede ver el diagrama de clases UML de aquellas clases que representan las relaciones.

Las clases del modelo presentadas hasta ahora tienen todas algo en común: trabajan con los datos de la aplicación que necesitan persistir en un soporte secundario, es decir, trabajan con datos que están en la base de datos MySQL. Con el objetivo de facilitar el posterior desarrollo de la aplicación, las clases del modelo presentadas no trabajan directamente con la base de datos, es decir, no lanzan ninguna operación de consulta sobre ella. En su lugar invocan a métodos de las librerías de acceso a datos. De esta manera, si se desea cambiar de tecnología de base de datos, basta con implementar unas nuevas librerías de acceso a datos, mientras que la lógica de la aplicación permanecerá intacta en los modelos. En el apartado 4.5.6 hablaremos sobre las librerías de acceso a datos.

En la figura 4.18 se pueden ver las clases que implementan el modelo de usuario. Los objetos de estas clases representan al usuario que está accediendo a la aplicación actualmente y contienen información acerca de éste. La mayor diferencia entre las dos clases que aparecen: Usuario-modelo y Admin-modelo, es que la primera carga los datos utilizando el modelo adecuado según el rol del usuario (alumo o profesor), mientras que la segunda no tiene soporte en base de datos.

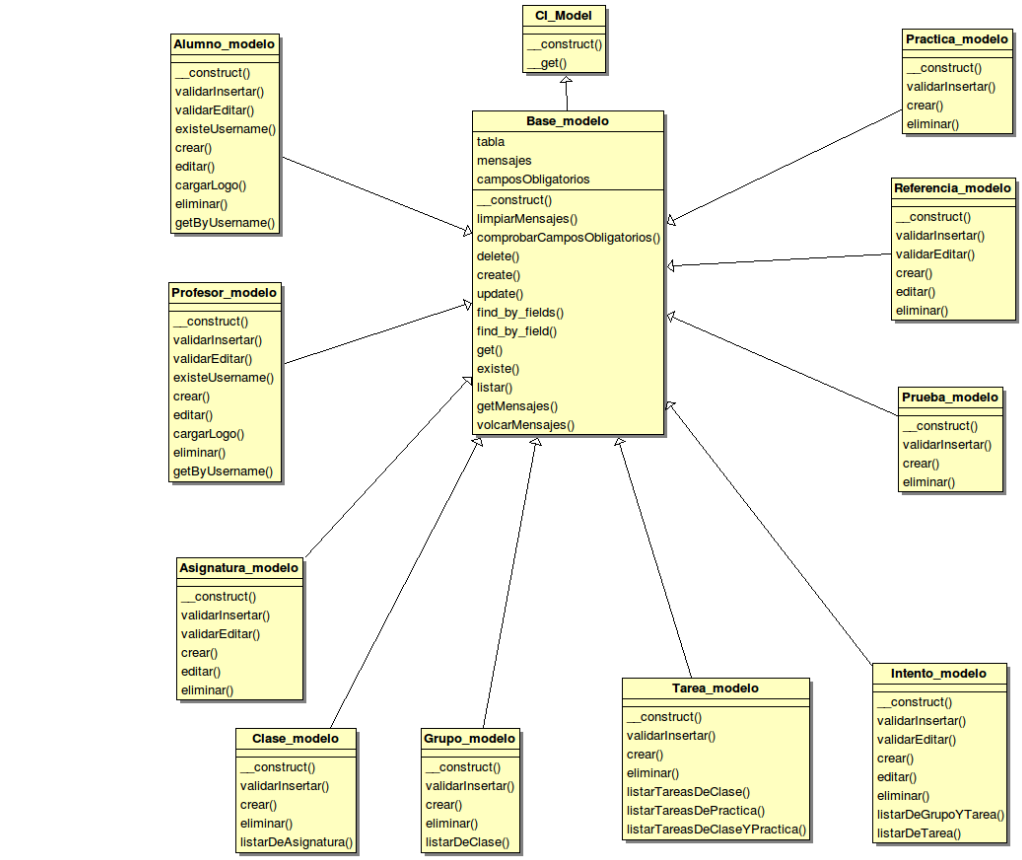


Figura 4.16: Diagrama de clases UML del modelo de datos I

4.5.4. Vistas

En el controlador base, del que hablaremos en la sección 4.5.5 dedicada a los controladores, se compone la estructura de la página. Todas las páginas del sitio siguen una estructura general organizada en cinco grandes secciones, identificables en la figura 4.14:

- **Cabecera:** contiene la cabecera HTML de la página.
- **Título:** contiene la barra superior con el logo y el título de la página.
- **Contenido:** esta vista es asignada en el método del controlador que se está invocando. Contiene el menú de acciones, el título descriptivo y el contenido específico de la página.
- **Menú lateral:** comprende el menú principal y la caja con información del usuario que está actualmente autenticado en la plataforma.
- **Pie:** contiene el pié de página.

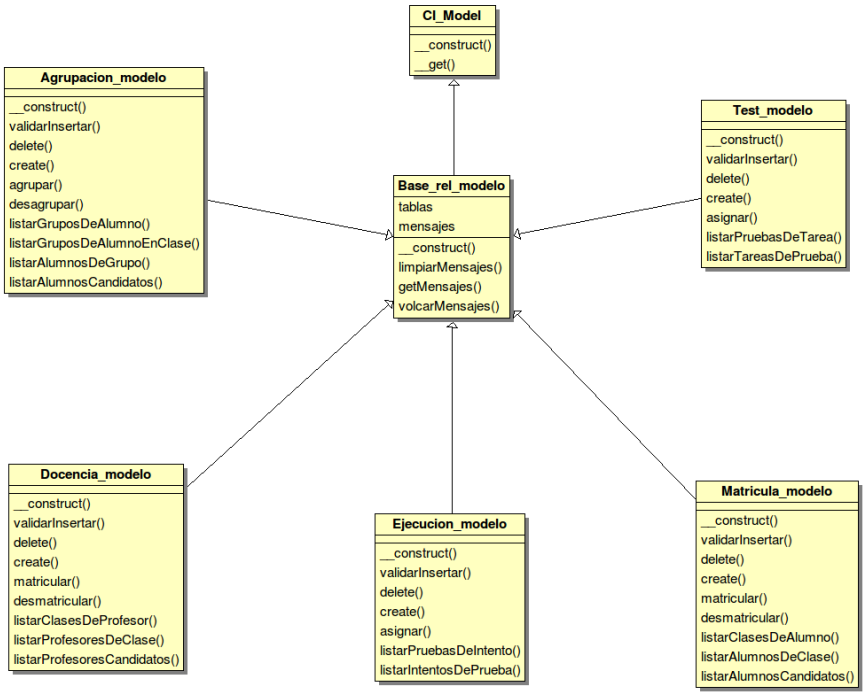


Figura 4.17: Diagrama de clases UML del modelo de datos II

Varias de estas secciones descritas anteriormente a su vez están divididas en dos o más vistas con el fin de facilitar su modificación para posteriores desarrollos. Los detalles de esta organización se describen en la documentación para desarrolladores de la que hablaremos en el apartado 4.6.2.

4.5.5. Controladores

Los controladores de la aplicación están organizados en dos grandes grupos: aquellos que implementan operaciones para alumnos y profesores, y aquellos que las implementan para administración. Los primeros heredan de la clase Base-controlador y los segundos de Base-admin-controlador.

Ambos controladores base siguen la misma filosofía: cualquier otro controlador de la aplicación se implementará como una clase que herede de un controlador base, el cual será responsable de:

- Cargar el modelo del usuario que actualmente esté accediendo a la aplicación, utilizando la capa de seguridad de CodeIgniter que se observa en la figura 4.15.

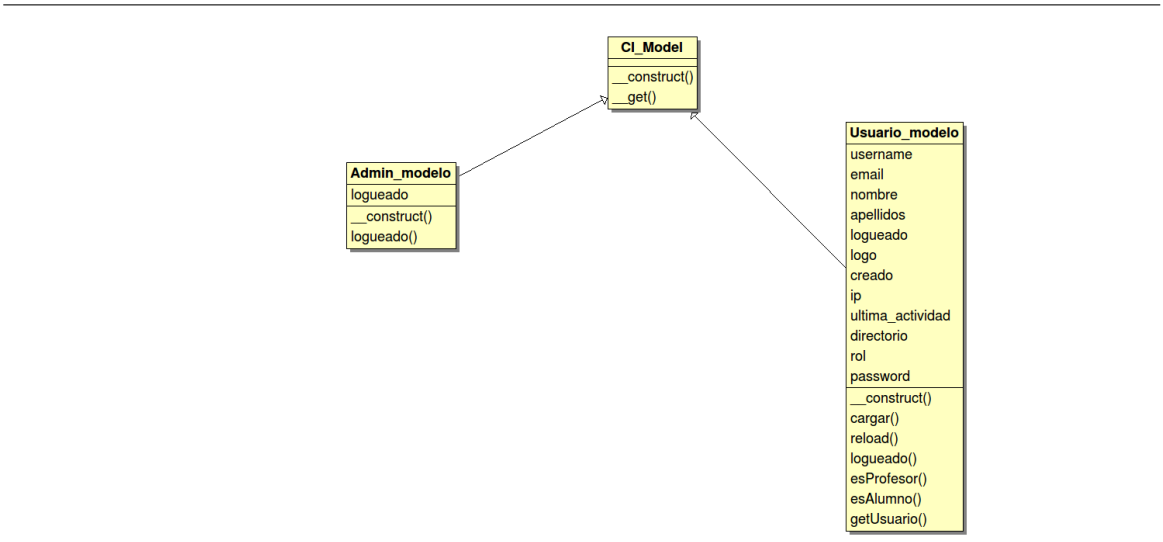


Figura 4.18: Diagrama de clases UML del modelo de usuario

- Componer la página HTML utilizando las vistas descritas en el apartado 4.5.4
- Proveer a las clases hijas de los métodos necesarios para construír el contenido dinámico de la página como, por ejemplo, el menú lateral, el título de la página, o los mensajes informativos como resultado de alguna operación.

En la figura 4.19 se pueden ver los controladores que componen la plataforma de administración y los métodos que implementan. Interesa la relación de herencia entre Base-admin y CI-Controller: CI-Controller es una clase que forma parte del núcleo de CodeIgniter, cualquier controlador de la aplicación debe heredar de ella.

Los controladores para alumnos y profesores siguen una organización muy similar a los de administración, como ya se ha explicado. En la figura 4.20 se describen las clases que constituyen los controladores para el acceso de alumnos.

Por último, en la figura 4.21 se muestran los controladores que constituyen la plataforma para profesores.

4.5.6. Librerías de acceso a datos

Como se ha comentado en el apartado 4.5.3, en las clases que implementan el modelo de datos no hay operaciones de consulta a un soporte específico como una base de datos, en su lugar hay llamadas a librerías auxiliares que se corresponden con operaciones básicas de alta, baja, modificación y consulta.

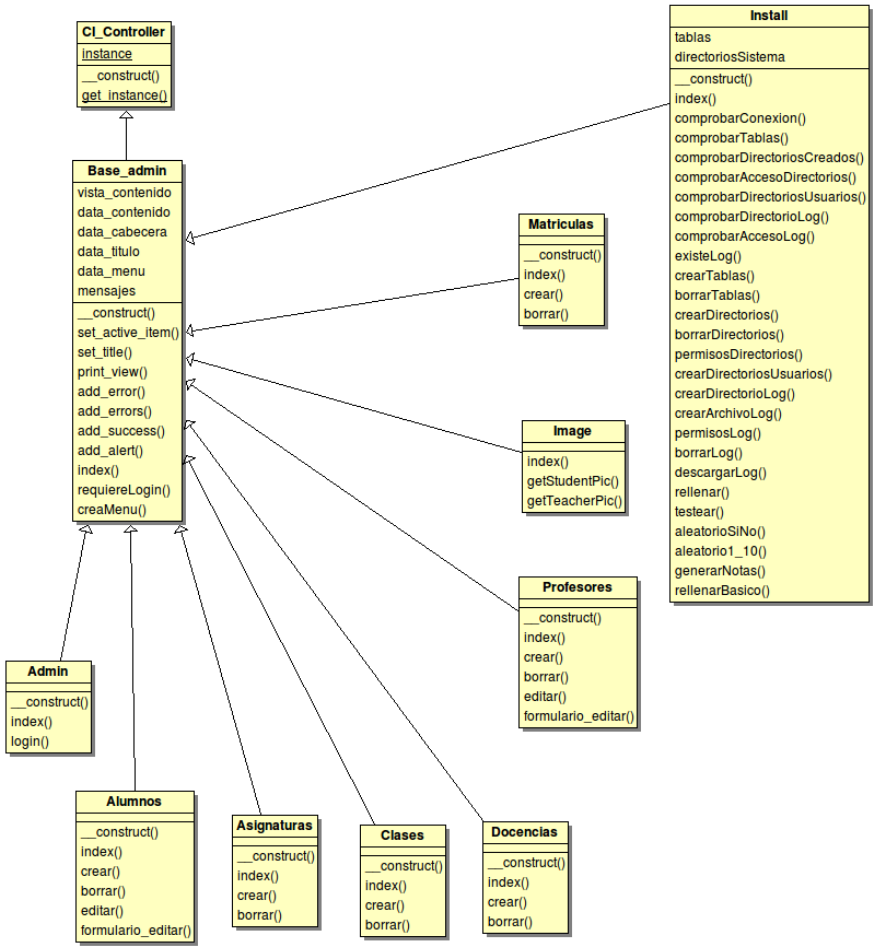


Figura 4.19: Diagrama de clases UML de los controladores del administrador

El objetivo de esta división de tareas es fomentar la portabilidad de la aplicación y su integración con otros sistemas de almacenamiento, así como facilitar la implementación aprovechando las características comunes de muchos de las entidades descritas en el citado apartado 4.5.3.

Las clases que aparecen en el diagrama de la figura 4.22 constituyen este conjunto de librerías que permite acceder a la base de datos MySQL que sigue el diseño explicado en el apartado 4.4.2. Interesa resaltar que la clase Datos parametriza el acceso a las tablas que representan entidades aprovechando el convenio que hemos adoptado de implementar las entidades como tablas cuya clave primaria sea un identificador autogenerado por la base de datos.

El resto de clases de la figura 4.22 proveen las operaciones de acceso a base de datos para

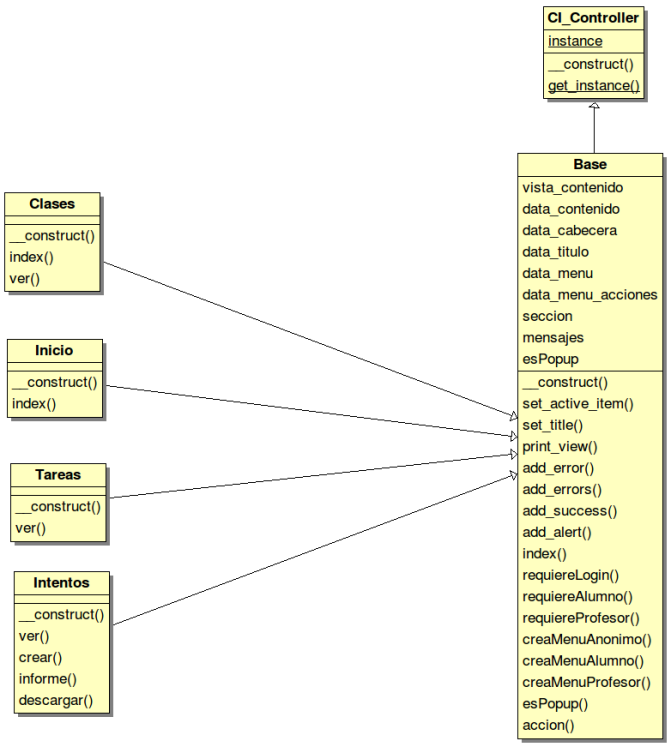


Figura 4.20: Diagrama de clases UML de los controladores de alumnos

las tablas que representan relaciones. Dado que los requisitos de acceso son más heterogéneos que en el caso de las entidades, y las operaciones de inserción requieren de una validación más compleja de los datos, no se han agrupado como en el caso anterior.

Por último, señalar que el atributo logger presente en todas las clases permite realizar un registro de las operaciones llevadas a cabo sobre la base de datos. En el apartado 4.5.7 se hablará más de este sistema de registro.

4.5.7. Integración con el resto del sistema

En este apartado describiremos cómo se integra la aplicación web con el resto del sistema descrito a lo largo de este documento. Las operaciones que permiten lograr esta integración se implementan en librerías.

En la figura 4.23 se pueden ver las dos clases que permiten la comunicación con el generador de estadísticas descrito en el apartado 3.2 y con el motor de pruebas descrito a lo largo del capítulo 3.

La clase Estadísticas que aparece en la figura 4.23 implementa diversos métodos que per-

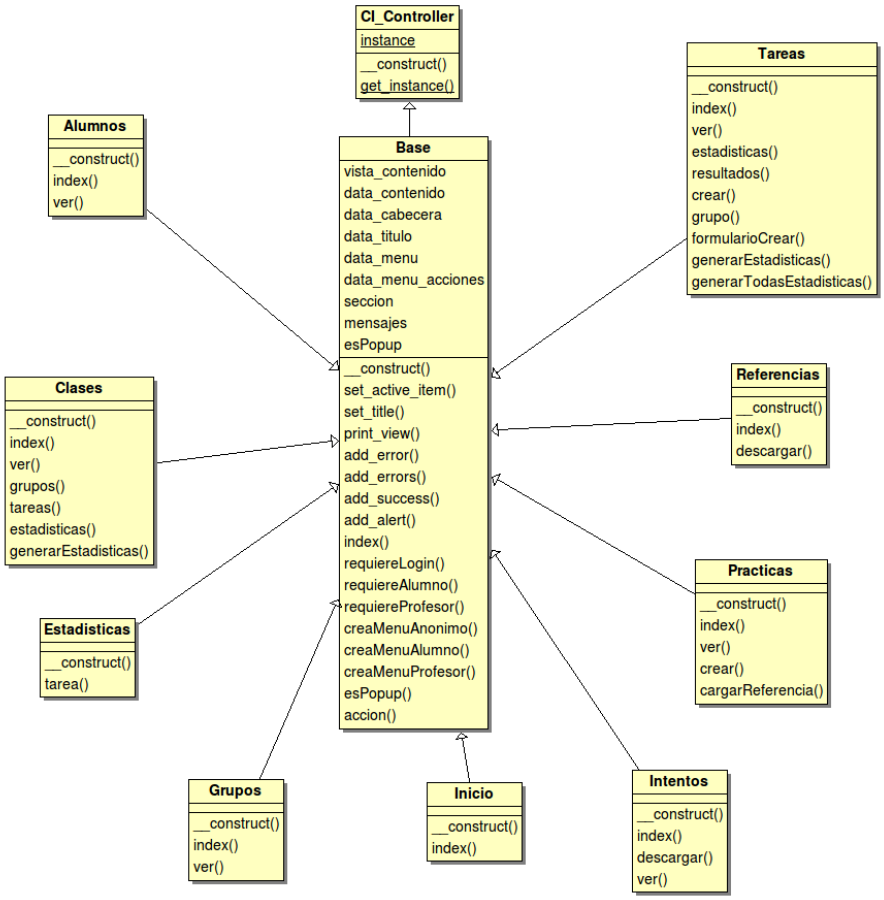


Figura 4.21: Diagrama de clases UML de los controladores de profesores

miten obtener las estadísticas generadas en forma de imágenes o bien solicitar la generación de alguna de estas estadísticas. Dicha solicitud se lleva a cabo invocando al propio programa del generador.

La clase ConexiónMotor que aparece en la figura 4.23 permite encolar nuevas solicitudes al motor de pruebas. Esta comunicación se lleva a cabo utilizando una tabla en la base de datos que actúa como cola. El motor irá atendiendo las solicitudes de esta tabla comenzando por la más antigua.

Figura 4.22: Diagrama de clases UML de las librerías de acceso a datos

4.6. Documentación

4.6.1. Manual de uso de la aplicación

Con el fin de facilitar el uso de la plataforma web hemos elaborado un completo manual de usuario. Está organizado en diferentes apartados, cada uno de los cuales ofrece información para alumnos, profesores y administradores de la plataforma. Para facilitar su acceso y su integración con la plataforma web, hemos elaborado dicho manual en formato Html.

El manual de usuario de la aplicación está disponible en la siguiente url: `http://projectx.fdi.ucm.es/web/manual/`.

4.6.2. Documentación para desarrolladores

La implementación de la plataforma cubre las funcionalidades más básicas del sistema informático descrito en la sección 4.2 titulada Especificación. Con el fin de promover su futuro desarrollo, hemos elaborado una completa documentación para desarrolladores que incluye información sobre la plataforma web que se ha descrito a lo largo del capítulo 4, sobre el generador de estadísticas presentado en la sección 3.2 y sobre el motor de pruebas expuesto a lo largo del capítulo 3.

La documentación para desarrolladores está redactada en formato Html y publicada junto

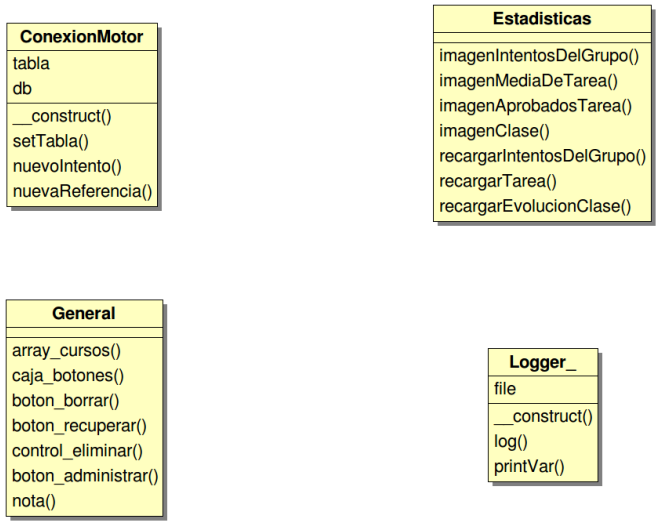


Figura 4.23: Diagrama de clases UML de las librerías de integración y registro de errores

con el manual de usuario en la url: `http://projectx.fdi.ucm.es/web/manual/`.

4.7. Mantenimiento

Las tareas de mantenimiento del sistema podrán ser llevadas a cabo por un administrador. Así pues, el usuario administrador tendrá acceso a un apartado en el cual podrá ver el estado actual de la base de datos del sistema así como de los directorios que utiliza para sus propios fines. En caso de que el sistema detecte algún problema, como puede ser la inexistencia de uno de los directorios o la imposibilidad de acceder, ofrecerá al administrador la posibilidad de intentar resolver el problema. En caso de tratarse de un problema de configuración, como puede ser la asignación de permisos a algunos directorios, mostrará la información oportuna que permita al administrador poder resolver el problema de manera manual.

La información ofrecida por el sistema se puede ver en la figura 4.24. En este caso no se ha detectado ningún problema.

Figura 4.24: Herramienta de mantenimiento

CAPÍTULO 5

Caso de estudio

Una vez diseñada la aplicación, decidimos probarla con datos reales. Estos datos reales proceden de una práctica asignada a los alumnos de Tecnología de la Programación, del Grado en Informática. Naturalmente nos fueron entregados previo filtro de datos confidenciales. Nuestros datos reales consisten en dos clases Java escritas por los alumnos:

- La clase `Numero`.
- La clase `Fecha`.

Podemos ver sendos diagramas UML de las clases en la figura 5.1.

La descripción de los métodos de la clase `Numero` es la siguiente:

- `esPrimo()` Devuelve True si el número es primo.
- `muestraDivisoresPrimos()` Muestra por pantalla todos los divisores primos de este número.
- `toString()` Devuelve un `String` describiendo la clase.
- `main(String[])` A los alumnos se le encargó escribir un método main que: creara un `Numero`, utilizara todos los métodos de la clase `Numero`.

La descripción de los métodos de la clase `Fecha` es la siguiente:

- `sumarDias(n)` Suma n días a la fecha indicada. Tomando en consideración que la fecha que calculamos debe ser una fecha correcta. Ej, si nuestra fecha es el 31-01-2001 y le sumamos 1 día, la nueva fecha sería 01-02-2001.

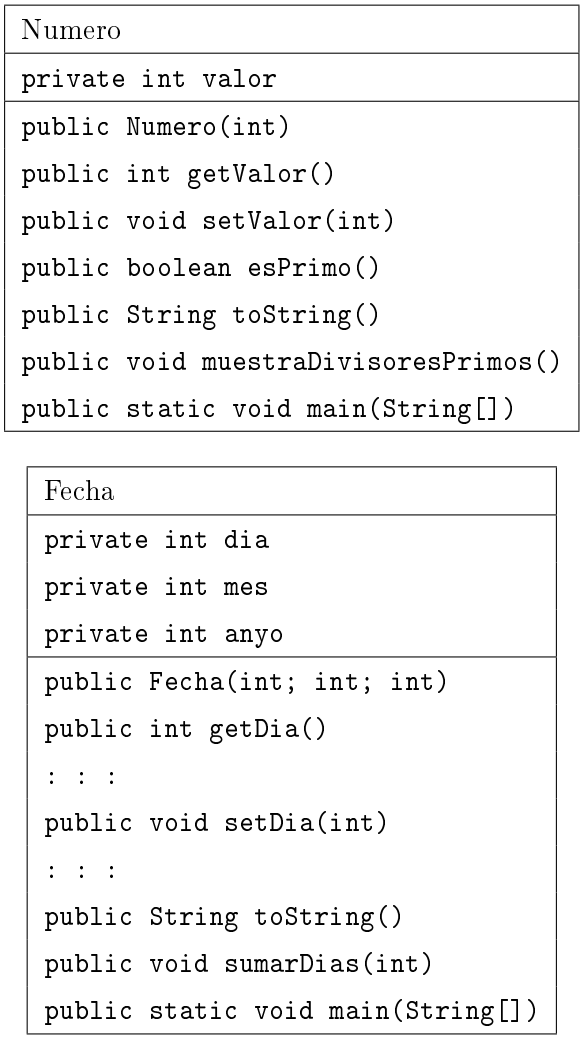


Figura 5.1: Diagrama UML clases `Numero` y `Fecha`.

- `toString()` Devuelve un `String` describiendo la clase.
- `public static void main(String[])` Este método deberá: crear una `Fecha`, utilizar todos los métodos de la clase `Fecha`.

Para ejecutar el caso de estudio necesitamos la referencia que proporcionaría el profesor. En este caso la componemos nosotros. Es importante reseñar que etiquetamos a los grupos con el identificador que nos fue proporcionado.

A continuación mostramos los resultados del caso de estudio:

5.1. Resultados

- `Grupo 183529`

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- `Grupo 188289`

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- `Grupo 149775`

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
	Total	10.0	
Exception	Numero	10.0	
	Total	10.0	
Correction	Numero	10.0	
	Total	10.0	
Performance	Numero	10.0	
	Total	10.0	

- Grupo 183852

Prueba	Clase	Calificación	Mensaje
Compare		0.0	
	Total	0.0	

- Grupo 183430

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 181970

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 188942

Prueba	Clase	Calificación	Mensaje
Compare	grupo14.hoja2.Numero	10.0	
Compare	grupo14.hoja2.Fecha	0.0	
	Total	5.0	

- Grupo 183850

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 182219

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 183115

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	0.0	
Performance	Fecha	10.0	
	Total	5.0	

- Grupo 180709

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
	Total	10.0	
Exception	Numero	10.0	
	Total	10.0	
Correction	Numero	10.0	
	Total	10.0	
Performance	Numero	0.0	
	Total	10.0	

- Grupo 178700

Prueba	Clase	Calificación	Mensaje
Compare	Fecha	10.0	
	Total	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 182472

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
	Total	10.0	
Exception	Numero	10.0	
	Total	10.0	
Correction	Numero	10.0	
	Total	10.0	
Performance	Numero	0.0	
	Total	0.0	

- Grupo 184950

Prueba	Clase	Calificación	Mensaje
Compare		0.0	
	Total	0.0	

- Grupo 181741

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	0.0	
	Total	5.0	

- Grupo 189214

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 178737

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 181377

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 182906

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
	Total	10.0	
Exception	Numero	10.0	
	Total	10.0	
Correction	Numero	10.0	
	Total	10.0	
Performance	Numero	0.0	
	Total	0.0	

- Grupo 187009

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 182655

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	0.0	
Performance	Fecha	10.0	
	Total	5.0	

- Grupo 179248

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 187674

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	0.0	
Performance	Fecha	10.0	
	Total	5.0	

- Grupo 146320

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	5.0	
Performance	Numero	10.0	
Performance	Fecha	0.0	
	Total	5.0	

- Grupo 179348

Prueba	Clase	Calificación	Mensaje
Compare	Fecha	10.0	
	Total	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Fecha	10.0	
	Total	10.0	

- Grupo 182623

Prueba	Clase	Calificación	Mensaje
Compare	Numero	10.0	
Compare	Fecha	10.0	
	Total	10.0	
Exception	Numero	10.0	
Exception	Fecha	10.0	
	Total	10.0	
Correction	Numero	0.0	
Correction	Fecha	0.0	La clase no contiene metodos aplicables.
	Total	0.0	
Performance	Numero	10.0	
Performance	Fecha	0.0	
	Total	5.0	

5.2. Interpretación de los resultados

A la vista de los resultados vistos, podemos sacar las siguientes conclusiones:

1. Algunos alumnos no han seguido el guión, creando las clases con nombres diferentes a los que se especifican. Esta circunstancia provoca que la herramienta no sea capaz de emparejarla con la referencia que el profesor proporciona. En este conjunto entrarían algunos grupos como el identificado como 184950, que sólamente proporcionó la clase Fecha pero la nombró 'Fechas'. Este obstáculo resulta representativo para la aplicación.

2. Otros alumnos no entregaron todas las clases solicitadas, sino solo la primera o la segunda.
3. Los alumnos que entregaron las clases siguiendo el formato del nombre, lo siguieron también con respecto a los métodos.
4. Algunos alumnos como el grupo 188942 utilizan su propia estructura de paquetes. Podemos comprobar que esto no es una dificultad para la aplicación.

`Nota:` Data la naturaleza de las clases y los métodos que se solicitan, los resultados de la prueba de corrección no se deben tener en cuenta. El motivo es que una de las clases solicitadas no tiene ningún método aplicable (en la clase `Fecha` solo se solicitan métodos `void`) y la otra tan solo cuenta con un método que no devuelve `void`, pero que depende de valores propios de su generación, que es arbitraria, y por tanto el resultado de la prueba para la clase `Numero` es arbitrario.

CAPÍTULO 6

Conclusiones y Trabajo Futuro

Es mejor ser aproximadamente correcto que precisamente erróneo

John Maynard Keynes

Principio de Incompatibilidad: "a medida que la complejidad de un sistema aumenta, disminuye nuestra capacidad para hacer afirmaciones precisas, incluso significativas, sobre su comportamiento, hasta que se alcanza un umbral más allá del cual precisión y relevancia son características casi mutuamente excluyentes"

Lofti Zadeh

6.1. Conclusiones

A lo largo del proyecto, hemos construído una herramienta con capacidad para ejecutar pruebas sobre clases sencillas en Java, gestionar datos tanto de profesores como de alumnos y elaborar estadísticas con ellos. Para ello hemos elaborado un pequeño framework estensible para desarrollar pruebas de forma sencilla, hemos implementado una plataforma que sirve de nodo para la comunicación alumno-profesores, y hemos definido una base de datos que almacena toda la información que se necesita en el proceso de manera adecuada.

En cambio no hemos llegado a dar el paso de la interpretación de los datos estadísticos. Este objetivo parecía de los más interesantes a priori, pero a lo largo del proyecto se fue diluyendo en favor de otras funcionalidades. En esta línea de trabajo cabe destacar nuestra intención de desarrollar un sistema que proporcione apoyo al profesor a la hora de calificar a los alumnos, dicho apoyo consitiría en lo siguiente: dados los resultados de una tarea, presentar al profesor a modo de sugerencia los pesos adecuados que se debería asignar a cada una de las pruebas. En este aspecto hemos pecado de ser demasiado ambiciosos, pues definir el concepto de "peso adecuado" no es un aspecto que resulte trivial de abordar. También hemos tenido que dejar como trabajo futuro algunas funcionalidades mencionadas en el apartado 4.1.3, más propias de un LMS[1], con el fin de centrarnos en otros aspectos de nuestro proyecto más innovadores.

Dentro de las decisiones que hemos ido tomando, consideramos un acierto la elección del cliente web como punto de interacción del sistema con los usuarios. La plataforma cubre todas nuestras necesidades y es sencilla de utilizar, sobre todo gracias a que no requiere ningún tipo de instalación por parte de los usuarios. Además, gracias al servidor que nos ha proporcionado el director del proyecto, hemos desarrollado nuestro trabajo en escritorio remoto con el apoyo de un repositorio de versiones (CVS), lo cual ha facilitado enormemente las tareas de coordinación del grupo. También consideramos acertado el enfoque para ejecutar pruebas. La Reflexión en Java nos da la posibilidad de crear pruebas genéricas para comprobar aspectos importantes en el desarrollo de ejercicios académicos.

Respecto a la plataforma, podríamos haber considerado su integración como un módulo de Moodle. Sin embargo, ni en la propuesta inicial ni a lo largo del desarrollo nos planteamos esa posibilidad. Sólamente cuando ya estaba la plataforma montada, consideramos esa opción. Como este proyecto, desde un principio, fue planteado como el desarrollo de una herramienta de apoyo a una asignatura, la integración en una plataforma campus virtual ya existente podría mejorar la usabilidad de la aplicación, acercando la herramienta

[1] Learning Management System

al usuario en un entorno que ya conoce y por tanto está acostumbrado a utilizar. Además, varios aspectos funcionales de la plataforma ya existen en Moodle (como el control de usuarios, la asignación de alumnos y profesores, la asignación de tareas...), con lo que nos habría facilitado también parte de este desarrollo.

A la hora de desarrollar la aplicación, hemos subestimado el handicap de la integración de componentes. Nunca antes habíamos gestionado la conexión de diferentes módulos implementados según distintos paradigmas y en diferentes lenguajes de programación, y finalmente debido a nuestra falta de previsión de este problema, se ha convertido en un retraso importante.

6.2. Trabajo futuro

Con el desarrollo de este proyecto, nos ha resultado llamativa la potencia de la Reflexión en Java para averiguar características de código en ejecución, y manipularlo como nos convenga. Con una perspectiva continuísta, consideramos clave seguir explotando la capacidad de la Reflexión. El techo hasta el que ampliar la construcción de pruebas se muestra lejano en este momento.

Así mismo, en el apartado anterior hemos comentado algunos aspectos en los que nos hubiese gustado indagar y no hemos podido por falta de tiempo. Un ejemplo es la interpretación no trivial de los resultados que ya somos capaces de obtener, con el objetivo de proporcionar información valiosa para estudiantes y profesores. Consideramos este un punto clave en el trabajo futuro sobre este proyecto, ya que está íntimamente ligado a nuestros objetivos iniciales y representa un amplio campo de trabajo.

Otro ejemplo es la citada integración de la plataforma como un módulo del campus virtual. Con vistas al futuro, sería una posibilidad a tener en cuenta

BIBLIOGRAFÍA

[1] M. Adler, M.K. Baumlin, and L.D. Richardson. Computer-assisted instruction. *Academic Emergency Medicine*, 7(12):1440–1440, 2000.

[2] R.M. Bottino. On-line learning networks: Framework and scenarios. *Education and Information Technologies*, 12(2):93–105, June 2007.

[3] J.R. Carbonell. Ai in cai: An artificial-intelligence approach to computer-assisted instruction. *Man-Machine Systems, IEEE Transactions on*, 11(4):190 –202, 1970.

[4] F. Di Cerbo, G. Dodero, and G. Succi. Extending moodle for collaborative learning. *SIGCSE Bull.*, 40(3):324–324, June 2008.

[5] S. Chiba. Load-time structural reflection in java. In *ECOOP 2000 — Object-Oriented Programming*, volume 1850 of *Lecture Notes in Computer Science*, pages 313–336. Springer, 2000.

[6] Plataforma educativa para los centros docentes de andalucía. `http://www.juntadeandalucia.es/averroes/helvia/sitio/`.

[7] A. Henrich and S. Sieber. Blended learning and pure e-learning concepts for information retrieval: experiences and future directions. *Information Retrieval*, 12(2):117–147, April 2009.

[8] Moodle.org: open-source community-based tools for learning. `http://moodle.org/`.

[9] Mysql :: The world's most popular open source database. `http://www.mysql.com/`.

[10] Oracle documentation. `http://docs.oracle.com/`.

[11] U. Rueda, M. Larra naga, A. Arruarte, and J.A. Elorriaga. Dynmap+: a concept mapping approach to visualize group student models. In *Proceedings of the First European conference on Technology Enhanced Learning: innovative Approaches for Learning and Knowledge Sharing,EC-TEL'06*, pages 383–397. Springer-Verlag, 2006.

[12] Sakai project | collaboration and learning - for educators by educators. `http://www.sakaiproject.org/`.

[13] L. Sheremetov and A.G. Arenas. Eva: an interactive web-based collaborative learning environment. *Computers & Education*, 39(2):161–182, September 2002.

[14] K.J. Sinclair, C.E. Renshaw, and H.A. Taylor. Improving computer-assisted instruction in teaching higher-order skills. *Computers & Education*, 42(2):169–180, 2004.

[15] H.J. So and T.A. Brush. Student perceptions of collaborative learning, social presence and satisfaction in a blended learning environment: Relationships and critical factors. *Computers & Education*, 51(1):318–336, August 2008.

[16] K. Stenning, J. McKendree, J. Lee, R. Cox, F. Dineen, and T. Mayes. Vicarious learning from educational dialogue. In *Proceedings of the 1999 conference on Computer support for collaborative learning, CSCL '99*. International Society of the Learning Sciences, 1999.

[17] G.T. Sullivan. Aspect-oriented programming using reflection and metaobject protocols. *Commun. ACM*, 44:95–97, 2001.

[18] B. Uğur, B. Akkoyunlu, and S. Kurbanoğlu. Students' opinions on blended learning and its implementation in terms of their learning styles. *Education and Information Technologies*, 16(1):5–23, March 2011.

[19] J.S. Vierling and M. Shivaram. On-line computer managed instruction: the first step. In *Proceedings of the November 17-19, 1970, fall joint computer conference, AFIPS '70*, pages 231–239. ACM, 1970.

[20] W3c: The world wide web consortium. `http://www.w3.org/`.